

А. А. Попов

ПРОГРАММИРОВАНИЕ

В СРЕДЕ

СУБД

FoxPro 2.0



«РАДИО И СВЯЗЬ»

А. А. Попов

**ПРОГРАММИРОВАНИЕ
В СРЕДЕ
СУБД
Fox Pro 2.0**

**ПОСТРОЕНИЕ
СИСТЕМ
ОБРАБОТКИ
ДАнных**



МОСКВА
«РАДИО И СВЯЗЬ»
1993

ББК 32.973

П41

УДК 681.3.06: 519.682.5

Редакция литературы по информатике и вычислительной технике

Попов А.А.

П41

Программирование в среде СУБД FoxPro 2.0. Построение систем обработки данных. — М.: Радио и связь, 1993. — 352 с.: ил.

ISBN 5-256-01170-7.

Рассматриваются вопросы программирования в среде одной из самых популярных СУБД для ПК — FoxPro-2.0. Описываются команды, составляющие язык, и способы их применения. Большое внимание уделяется созданию пользовательского интерфейса. Разбираются типовые задачи, возникающие при создании прикладных систем обработки данных (ведение баз данных, работа со словарями, древовидная организация данных и т.п.), рассматриваются средства автоматизации программирования — генераторы экранов, отчетов, меню и проектов.

Книга ориентирована на самостоятельное и активное овладение техникой программирования прикладных систем обработки данных и может быть полезной для любых категорий читателей — студентов, начинающих программистов в среде FoxPro и опытных разработчиков.

2404040000-082

П ----- Без объявл.
046(01)-93

ББК 32.973

ИБ № 2578

ISBN 5-256-01170-7

© Попов А.А., 1993

ВВЕДЕНИЕ

Опыт применения ЭВМ для построения прикладных систем обработки данных показывает, что самым эффективным инструментом здесь являются не универсальные алгоритмические языки высокого уровня, а специализированные языки для создания систем управления данными. Такие средства обычно включаются в состав систем управления базами данных (СУБД), но они могут существовать и отдельно. СУБД дают возможность пользователям осуществлять непосредственное управление данными, а программистам быстро разрабатывать более совершенные программные средства их обработки. Характеристики готовых прикладных пакетов определяются прежде всего принятой в СУБД организацией данных и типом используемого транслятора.

Модели данных. По способу установления связей между данными различают реляционную, иерархическую и сетевую модели.

Реляционная модель является простейшей и наиболее привычной формой представления данных в виде таблицы. В теории множеств таблице соответствует термин *отношение* (relation), который и дал название модели. Для нее имеется развитый математический аппарат — реляционное исчисление и реляционная алгебра, где для баз данных (отношений) определены такие хорошо известные теоретико-множественные операции, как объединение, вычитание, пересечение, соединение и др.

Достоинством реляционной модели является сравнительная простота инструментальных средств ее поддержки, недостатком — жесткость структуры данных (невозможность, например, задания строк таблицы произвольной длины) и зависимость скорости ее работы от размера базы данных. Для многих операций, определенных в такой модели, может оказаться необходимым просмотр всей базы.

Иерархическая и сетевая модели предполагают наличие связей между данными, имеющими какой-либо общий признак. В иерархической модели такие связи могут быть отражены в виде дерева-графа, где возможны только односторонние связи от старших вершин к младшим. Это облегчает доступ к необходимой информации, но только если все возможные запросы отражены в структуре дерева. Никакие иные запросы удовлетворены быть не могут.

Указанный недостаток снят в сетевой модели, где, по крайней мере теоретически, возможны связи "всех со всеми". Поскольку на практике это, естественно, невозможно, приходится прибегать к некоторым ограничениям. Использование иерархической и сетевой моделей ускоряет доступ к информации в базе данных. Но поскольку каждый элемент данных должен содержать ссылки на некоторые другие элементы, требуются значительные ресурсы как дисковой, так и основной памяти ЭВМ. Недостаток основной памяти, конечно, снижает скорость обработки данных. Кроме того, для таких моделей характерна сложность реализации СУБД.

СУБД для персональных компьютеров. Хотя известны попытки создания систем управления базами данных, поддерживающих сетевую модель для персональных компьютеров, в настоящее время реляционные системы лучше соответствуют их техническим возможностям и вполне удовлетворяют большинство пользователей. Скоростные характеристики этих СУБД поддерживаются специальными средствами ускоренного доступа к информации — индексированием баз данных.

Прежде чем перейти к рассмотрению конкретных пакетов, уместно уточнить само понятие системы управления базами данных. В наиболее полном варианте такой пакет может иметь следующие компоненты:

Среда пользователя, дающая возможность непосредственного управления данными с клавиатуры.

Алгоритмический язык для программирования прикладных систем обработки данных, реализованный как интерпретатор. Последнее позволяет быстро создавать и отлаживать программы.

Компилятор для придания завершенной программе вида готового коммерческого продукта в форме независимого EXE-файла.

Программы-утилиты быстрого программирования рутинных операций (генераторы отчетов, экранов, меню и других приложений).

Собственно СУБД — это, конечно, оболочка пользователя. Ввиду того что такая среда ориентирована на немедленное удовлетворение его запросов, это всегда система-интерпретатор. Есть множество хороших зарубежных пакетов, которые имеют только один указанный компонент. Однако для отечественного пользователя он представляет наименьшую ценность, поскольку, как показывает опыт, трудности овладения англоязычным интерфейсом быстро отпугивают потенциальных потребителей.

Наличие в СУБД языка программирования позволяет создавать сложные системы обработки данных, ориентированные под конкретные задачи и даже под конкретного пользователя. Есть также СУБД, которые имеют только язык и не имеют оболочки пользователя. Они предназначены исключительно для программистов, и это системы компилирующего типа. Такие пакеты лишь с оговорками могут быть названы СУБД. Обычно их называют просто компиляторами.

Группа реляционных СУБД представлена на рынке программных продуктов очень широко. Это, например, такие системы, как Paradox, R:base, Clarion, однако доминирующее положение здесь занимает семейство так называемых dBASE-подобных СУБД, родоначальником которого является СУБД dBASEII, предложенная фирмой Ashton-Tate в начале 80-х годов. Система и в особенности следующие ее версии оказались столь популярными, что появилось несколько фирм-последователей, которые позаимствовали не только идеологию Ashton-Tate, но в значительной степени и ее язык. Вместе с тем предлагаемые ими системы являются лицензионно-чистыми, во многом оригинальными и часто более эффективными. В это семейство кроме самой СУБД dBASE входят системы управления базами данных: FoxBASE, Clipper, Quick Silver, DBXL, Dbfast (здесь перечислены как настоящие СУБД, так и только компиляторы). Кроме того, на рынке программных продуктов имеется большое количество различных вспомогательных программ-утилит, облегчающих выполнение тех или иных задач, возникающих при работе с системами такого типа.

Из перечисленных три последние СУБД у нас практически не известны, но зато остальные широко распространены и даже были русифицированы. Это следующие версии:

dBASEIII-plus (русифицированная версия РЕБУС),
FoxBASE-plus версия 2.0 и 2.1 (КАРАТ/М-2.0 и 2.1),
Clipper-87 (СуперКом-87).

В настоящее время широко распространено новое поколение этих популярных пакетов: dBASEIV, FoxPro и Clipper-5.

Важнейшей характеристикой любой СУБД является используемый в ней тип транслятора (интерпретатор или компилятор). Программы, написанные для системы-интерпретатора, исполняются лишь в присутствии самой системы. В настоящее время скорость работы таких программ не уступает скорости программ, сгенерированных компилятором. Бесспорным преимуществом интерпретаторов для программиста является удобство в разработке и отладке программных продуктов, а также при освоении языка. Из

вышеперечисленных СУБД dBASE и FoxPro являются интерпретаторами, а Clipper — компилятором.

Фирма Ashton-Tate представила радикально модернизированную по сравнению с dBASEIII-plus СУБД dBASEIV. В ней еще более развиты средства непосредственного доступа к данным, в том числе реализован распространенный на более мощных ЭВМ язык управления запросами SQL, добавлено много новых команд и функций. Однако сама среда получилась очень громоздкой, что определило низкую скорость ее работы на ПЭВМ обычной конфигурации. Пакет оказался неудачным, и фирма потеряла ведущее положение на рынке СУБД. Это явилось одной из причин поглощения ее концерном Borland. Последний продолжает работу над системой и выпустил новую версию пакета — dBASEIV-1.5, в которой устранены многие недостатки предшественницы. Однако по отзывам печати система остается очень медленной.

Первые версии системы Clipper (фирма Nantucket) были просто компиляторами к СУБД dBASE. Затем она вместе со своим программным окружением превратилась в полноценную и независимую языковую среду для построения систем обработки данных. После компиляции программ, созданных с помощью Clipper, формируются загрузочные модули типа EXE, которые далее могут запускаться самостоятельно без поддержки их "родительской" СУБД, как это имеет место в системах-интерпретаторах. Недостатком систем-компиляторов являются большие суммарные затраты времени на многократную компиляцию и сборку ("линковку") исходных модулей программ при ее отладке, что очень замедляет труд разработчика.

СУБД FoxPro (фирма Fox Software) обладает исключительно высокими скоростными характеристиками и в этом отношении заметно выделяется среди интерпретирующих систем. Сравнительно с dBASEIV ее скорость в несколько раз выше и не уступает скорости систем-компиляторов. Практически по всем показателям Фокс-программы работают значительно быстрее Clipper-программ. Набор команд и функций, предлагаемых разработчикам программных продуктов в среде FoxPro, по мощи и гибкости отвечает любым современным требованиям к представлению и обработке данных. Здесь может быть реализован максимально удобный, гибкий и эффектный пользовательский интерфейс. В FoxPro поддерживаются разнообразные всплывающие и многоуровневые меню, работа с окнами и мышью, реализованы функции низкоуровневого доступа к файлам, управление цветами, настройка принтера, данные могут быть представлены в виде, похожем на электронные таблицы, и т.п. Система также обладает средствами быстрой генерации экранов, отчетов и меню, поддерживает язык SQL, хорошо работает в сети. В пакете имеется компилятор, позволяющий при желании сформировать EXE-файлы готовых программ.

FoxPro получил прекрасную прессу на Западе. В настоящее время это, несомненно, самый быстрый пакет среди СУБД для персональных компьютеров стандарта IBM PC.

Переломными в коммерческой судьбе dBASE-СУБД оказались 1991-1992 гг. Фирмы-разработчики систем dBASE, Clipper и FoxPro слились с крупными концернами-изготовителями программной продукции — Borland, Computer Associates (хорошо известна у нас по электронной таблице Supercalc-4 и 5) и Microsoft соответственно. Причем для фирм Ashton-Tate и Nantucket этот шаг был вынужденным ввиду с затруднениями при распространении пакетов dBASEIV и Clipper-5 и напряженным финансовым положением. В случае с Fox Software ситуация обратная. Компания Microsoft пошла на приобретение Fox Software, учитывая высокий рейтинг и перспективность СУБД FoxPro. В связи с этим следует ожидать, во-первых, исключительного усиления позиций FoxPro на мировом рынке и его быстрого развития, во-вторых, официального

появления, адаптации и сопровождения пакета FoxPro в России, поскольку Microsoft активно работает в стране, и вообще занятия им достойного места среди инструментальных средств для ПК. Ведь до сих пор, несмотря на общепризнанные достоинства пакета, он имеет непропорционально малую часть рынка систем управления базами данных, существенно отставая от лидера — СУБД Paradox фирмы Borland.

Работая над книгой, автор столкнулся с невозможностью "гладкого" изложения материала "по нарастающей", когда он предлагается строго последовательно от простого к сложному с постепенным и дозированным повышением трудности. Язык FoxPro настолько богат возможностями, что уже при изучении первых основных команд (например, команды BROWSE) требуются понятия (индексы, функции, шаблоны и т.д.), которые еще только предстоит ввести. Такие случаи, обычно встречающиеся в примерах, по-возможности комментируются на месте. Неясные места пока можно пропускать либо, забегая вперед, обращаться к соответствующим разделам.

При этом не следует слепо полагаться на эту или какую-то другую книгу или даже на официальное техническое описание СУБД, предлагаемое фирмой-разработчиком. Расхождение между словом и фактом может возникнуть как вследствие ошибки в документации или в пакете, так и ввиду того, что фирмы объявляют (иногда авансом) присутствующими возможности языка, которые не были полностью реализованы к моменту кодирования пакета; и наконец, обратная ситуация — пакет модернизирован, а описание еще нет. Несомненно также, что к моменту выхода книги появятся новые команды и функции, а также, возможно, новые виды данных, новые режимы работы, новые утилиты, изменятся количественные характеристики, некоторые детали интерфейса и т.п., ведь FoxPro — очень динамичная система. Вместе с тем работа приводимых в книге команд и функций, а также все программы практически проверены на компьютере.

Любая мощная программная среда (СУБД, электронная таблица, издательская система, развитый язык высокого уровня), каковой, безусловно, можно считать FoxPro, является по существу целой страной, где могут быть и темные места. Все это мы подчеркиваем для программистов, только начинающих изучение СУБД. Разработчики, имеющие даже небольшой опыт, и так это прекрасно знают.

Продуктивное освоение языка FoxPro реально, конечно, только при практической проработке материала на компьютере. Используя командный и программный режимы, следует по возможности опробовать все команды, функции и предлагаемые программы. Вообще и в дальнейшем все сомнительные или неясные команды и функции следует просто проверить самому. Ведь это так легко сделать в среде-интерпретаторе — результат получается немедленно, особенно в командном режиме.

В некоторых приводимых примерах, чтобы подчеркнуть, что данная команда выполняется здесь в командном режиме, т.е. вводится с клавиатуры в окне Command, ей в тексте будет предшествовать знак — точка (.). Для всех пользователей систем dBASE и FoxBASE — это хорошо знакомое приглашение к вводу в командной строке. В FoxPro никакой точки в командном окне нет, и тем более не нужно вводить ее самим. Она является только лишь обозначением командного режима в тексте книги.

В предлагаемом материале по FoxPro не ставится задача детального разбора всех более чем пяти сот ее команд, функций и системных переменных. Эти сведения можно найти в технической документации. Главное внимание уделено более важному аспекту — изучению техники программирования в среде СУБД FoxPro.

Глава 1. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ И ОСОБЕННОСТИ СУБД

Для функционирования СУБД FoxPro необходимо наличие жесткого диска и минимального объема памяти персональной ЭВМ 512 Кбайт (или 420 Кбайт свободной памяти). Полный комплект FoxPro-2.0 состоит из следующих компонентов:

Собственно СУБД (Data Base Managment System — DBMS).

Дистрибутивный пакет (Distribution Kit), предназначенный для распространения готовых программных продуктов. Пакет содержит компилятор для создания автономных EXE-файлов, а также исполнительную систему (RunTime) в виде специальных библиотек и средств формирования неавтономных EXE-файлов для работы в ее среде.

Пакет разработки собственных библиотек (Library Construction Kit) на языках Си и Ассемблер с использованием нового Интерфейса Прикладных Программ (Application Program Interface — API). Наличие этого компонента делает FoxPro открытым для дополнения и модернизации, благодаря чему появилось большое число библиотек различных фирм для деловой и иллюстративной графики, телекоммуникаций, работы с серверами баз данных, сетями.

FoxPro распространяется сразу в двух версиях — стандартной (для ПК с процессором i8088/86 и выше) и расширенной (i80386/486). При инсталляции системы компоненты, относящиеся к расширенной версии, помечены символом (X). Главные файлы такой версии также имеют букву X в конце имени.

Кроме того, FoxPro продается как в сетевом (FoxPro/LAN), так и в обычном (несетевом) варианте. Главные файлы сетевой версии имеют букву L в конце имени. Все перечисленные варианты пакета имеют некоторые особенности, которые мы здесь не рассматриваем.

При инсталляции стандартной несетевой версии FoxPro в главную директорию СУБД (обычно с именем FOXPRO2) копируются следующие основные группы файлов:

1. FOXPRO.EXE, FOXPRO.OVL — среда пользователя и разработчика.
2. CONFIG.FP — файл конфигурации FoxPro.
3. FOXUSER.DBF, FOXUSER.FPT — файлы внутренних установок FoxPro.
4. FOXHELP.DBF, FOXHELP.FPT — HELP к системе FoxPro.
5. FOXDOC.EXE, FOXDOC.MSG, FOXDOC.HLP, FOXDOC.OVR, CONFIG.FXD, PROWORDS.FXD — файлы Документатора FoxPro (Fox-Doc).
6. GENMENU.PRG — генератор программных кодов меню, созданных построителем меню.
7. GENSCRN.PRG — генератор программных кодов экранов, созданных построителем экранов.
8. FOXSWAP.COM — программа для организации взаимодействия СУБД FoxPro с внешними программами, загруженными по команде RUN. В расширенной версии не используется.
9. GENPD.APP — генератор программ поддержки принтеров.
10. DEMO.APP, DEMO.BAT — демонстрационный "ролик" системы.
11. FOX.EXE, FOXS.EXE/FOXLEXE — программы-загрузчики FoxPro. С их помощью можно загрузить нужный вариант системы. Использование загрузчиков может быть оправдано в сетевом варианте, когда компьютеру доступны сразу несколько версий (сетевая, несетевая, стандартная, расширенная) FoxPro. По умолчанию загрузчик вызывает

"максимальную" версию. С помощью вводимых ключей можно загрузить и конкретную версию пакета. Поскольку на винчестере имеется обычно только один вариант системы, применение загрузчиков не имеет смысла в однопользовательском режиме. Кроме того, будучи использованным, загрузчик далее занимает некоторый объем основной памяти. Вызов системы лучше осуществлять вызовом ее головного файла FOXPRO.EXE.

Среди перечисленного совершенно обязательными являются только два первых главных файла (п.1), хотя обычно всегда присутствуют и файлы из пп. 2, 3, 4.

При инсталляции FoxPro может быть создано множество директорий, содержащих учебные и вспомогательные файлы программ, данных, экранов, меню и т.п. Этот материал может понадобиться при освоении системы.

Особенности языка FoxPro

Прежде чем перейти к освоению собственно языка, уместно хотя бы кратко перечислить его новые возможности, важные для программиста.

- Динамическое распределение памяти компьютера. Системе доступна память за пределами 640 Кбайт. Стандартный вариант FoxPro дает возможность использовать Expanded LIM-4.0 память. Причем 64 Кбайта может быть задействовано для хранения окон и других данных, а остальная часть — под весьма эффективный буфер ввода-вывода. Расширенная версия работает в защищенном режиме и использует всю Extended-память, установленную на машине (Expanded-память не используется).

- Система "замечает" имеющийся сопроцессор. При этом ее характеристики существенно улучшаются.

- Поддержка и активное использование мыши.

- Язык FoxPro позволяет создавать хорошо структурированные программы. Практически из всех опций команд могут быть вызваны пользовательские процедуры/функции, что придает ему исключительную гибкость. Допускаются как внешние, так и внутренние процедуры.

Особенно необходимо указать, что теперь отпала требующая большой внимательности разработчика необходимость в программной организации двух самых типичных циклов, всегда присутствующих в прикладных системах обработки данных.

Во-первых, это цикл регенерации меню, поскольку завершение обработки любого выбора пользователя из меню обычно требует обратного возврата в него.

Во-вторых, поддержка окна редактирования текущей записи базы данных. Так как редактирование подразумевает перемещения в базе и другие трансформации данных, необходимо постоянно иметь возможность покидать и возвращаться в это окно с его регенерацией командой READ, после чего возможно выполнение следующих действий.

В FoxPro такие циклы теперь могут быть реализованы автоматически. Программист больше о них не заботится.

- Средства удобного доступа и обработки мемо-полей. Теперь они могут использоваться практически наравне с символьными полями.

- Новые типы индексных файлов, а также возможность применения индексов не только для поиска отдельной записи, но и быстрой локализации групп записей с общим признаком (технология Rushmore).

- Установка реляционных связей между базами данных вида одна_запись-ко-многим.

○ Возможность построения многоуровневых световых меню, в качестве элементов которых могут фигурировать как произвольные строки, так и элементы массивов и компоненты файлов. Допускается множественный отбор данных. Меню получили статус объектов в памяти компьютера.

○ СУБД предоставляет программисту средства конфигурации и системного меню самого FoxPro, что дает возможность легко настраивать его для работы в оболочке FoxPro (например, при отладке) и даже интегрировать в прикладную программу.

○ Широкое использование концепции окон, допускающее многооконный интерфейс, изменение пользователем размера, вида и положения окон, управление цветом.

○ FoxPro стал располагать средствами языка SQL, дающими программисту исключительные возможности по формированию сложных запросов к базе, обрабатываемых с использованием одного из самых интеллектуальных оптимизаторов запросов.

○ Специализированные команды обработки массивов.

○ Низкоуровневый доступ к файлам.

○ Удобный и мощный отладчик.

○ Язык FoxPro включает теперь средства создания Windows-подобного интерфейса, т.е. так называемого интерфейса, управляемого событиями.

Все привычные нам до сих пор прикладные системы обработки данных имеют более или менее жесткий сценарий, допускающий лишь небольшую свободу пользователя по управлению данными, когда для принятия и реализации того или иного решения ему приходилось осуществлять иногда довольно длительный спуск/подъем по системам меню и окнам. При этом он часто забывал, как и зачем пришел в данное место. Кроме того, количество одновременно доступных пользователю объектов очень невелико, а облик интерфейса для любого состояния системы фиксирован.

Теперь на одном экране можно разместить сразу много разнообразных окон редактирования и средств управления данными, все они легко доступны пользователю. Именно здесь эффективно использование мыши.

Такой экран является наиболее полным аналогом рабочего стола конторского работника, иногда в беспорядке заваленного горами документов, картотеками, справочниками и т.п., которые могут понадобиться ему в любой момент. FoxPro предлагает интерфейс, обеспечивающий быстрый доступ ко всем нужным данным и вместе с тем возможность поддержания их четкой взаимосвязи и порядка. Если оказывается, что на экране-столе слишком тесно, временно ненужные документы могут быть отодвинуты, уменьшены или даже совсем "убраны" в сторону (временно скрыты с экрана), или отображены только своим заголовком с возможностью раскрыть их при необходимости. Программирование такого интерфейса не является тривиальным делом. Разработчику следует позаботиться о правильном интегрировании всех экранных объектов, а также контролировать свое положение среди них. Эту технологию предстоит еще только осознать.

Вообще придание меню статуса объектов и введение Windows-подобных средств управления коренным образом меняет концепцию программирования систем обработки данных, хотя остается доступной возможность работы и в традиционном стиле. Кроме того, FoxPro имеет в своем составе такие важные вспомогательные средства, как генераторы экранов, отчетов, меню; документатор программ; менеджер проектов и компилятор; драйверы принтера, т.е. все то, что делает язык СУБД полнокресным языком четвертого поколения. Поскольку сами генераторы написаны на языке FoxPro, они открыты для любых модификаций по вкусу программиста.

Стоит указать, что FoxPro дает разработчику (в отличие от подавляющего

большинства других пакетов) возможность самому написать удобный драйвер принтера.

Совершенно необходимо также отметить API FoxPro. Он исключительно развит (по сравнению, например, с Clipper-5) и предоставляет программисту возможность доступа к структурам самого FoxPro — переменным, меню, окнам. Более того, теперь можно встроиться в очередь обработки событий FoxPro или в очередь процедур, выполняемых в то время, пока FoxPro ожидает действий пользователя. Это, например, дает возможность в фоновом режиме осуществлять телекоммуникационный сеанс. Предоставляется также возможность низкоуровневого доступа к файлам с использованием высокоэффективных алгоритмов кеширования самого FoxPro.

Автор хотел бы заметить, что практическая работа с пакетом дает ежедневное подтверждение правильности выбора инструмента. Более того, он впервые ощутил совершенно новое качество — возможность реализовать любые собственные идеи и все, иногда взыскательные, пожелания заказчиков по организации обработки данных и интерфейсу пользователя.

Терминология и структура данных в СУБД

Все данные и другая информация СУБД хранятся на магнитных дисках в дисковых файлах. Файл данных, или база данных, представляет собой таблицу (рис.1.1), каждая строка которой (запись) содержит некоторые сведения об описываемом объекте/объектах. Каждая клетка записи называется полем записи. Все записи базы данных имеют идентичную, заданную пользователем структуру и размеры.

База данных				
Поле 1	Поле 2	...	Поле N	Запись 1
		...		Запись 2
		...		Запись K

Рис.1.1

Так, записи базы Кадры могут содержать, например, поля: фамилия, год рождения, семейное положение и др. Единицей длины данных является байт — 8 двоичных разрядов. Обычно один символ занимает один байт.

Типы файлов

В FoxPro можно создавать и обрабатывать несколько типов дисковых файлов. Каждому файлу программист дает имя по обычным правилам, принятым в MS DOS, т.е. собственно имя длиной до восьми символов и (через точку) расширение длиной до трех букв:

<ИМЯ ФАЙЛА>.<РАСШИРЕНИЕ>

Расширение определяет тип файла. Для всех файлов FoxPro установлены стандартные расширения. Вот основные типы файлов:

- | | |
|-----------|--|
| <имя>.DBF | — файл базы данных; |
| <имя>.FPT | — файл примечаний, хранящий мемо-поля базы данных; |
| <имя>.IDX | — индексный файл; |
| <имя>.CDX | — мультииндексный файл; |
| <имя>.PRG | — командный, программный файл; |
| <имя>.MEM | — файл для сохранения временных переменных; |
| <имя>.FXP | — откомпилированный командный файл PRG. |

Файлы DBF являются главными файлами данных в СУБД и термин "база данных" будем относить именно к ним.

Замечание. Не разрешается DBF-файлам давать односимвольные имена из букв от А до J, так как эти имена зарезервированы под рабочие области, в которых располагаются файлы баз данных.

Файлы базы данных DBF и FPT

Файлы БД типа DBF являются основными носителями данных на диске. Они имеют следующие характеристики:

число записей в файле	— до 1 млрд;
размер записи (в байтах)	— до 4000;
число полей в записи	— до 255;
число одновременно открытых баз	— до 25.

Типы и размеры полей (в байтах):

символьные поля	— до 254;
числовые поля	— до 20;
поля дат	— 8;
логические поля	— 1.

Символьные поля (поля типа C) допускают ввод любых алфавитно-цифровых символов, знаков препинания и т.д.

Числа в базе данных хранятся в числовых полях двух форматов — с фиксированной и плавающей точками (типы N и F).

При фиксированном (естественном) представлении под числовое поле, включая целую и дробную части, десятичную точку и знак минус (если есть), может быть занято от 1 до 20 разрядов. Целое число может занимать все 20 разрядов, дробное — от 0 до 18 разрядов и один разряд под точку и возможный знак "+" или "-".

Однако достоверными при вычислениях в FoxPro являются только 16 разрядов.

Формат числа с плавающей точкой здесь не отвечает общепринятому смыслу этого термина и ничем не отличается от формата с фиксированной точкой. Он введен для совместимости с dBASEIV, и, как указано в документации, его удобно использовать в научных расчетах. Чем именно, остается загадкой. Возможно этот формат будет реализован позже.

Логические поля (тип L) и другие логические величины могут иметь только два значения: .T. (от TRUE — "Истина") и .F. (от FALSE — "Ложь") или, что то же самое, .Y. и .N. (от YES — "Да" и NO — "Нет"). Обрамляющие точки вводить не нужно. Разрешается использовать и строчные буквы.

Поля дат (тип D) допускают ввод, естественно, только цифр. Разрешенные даты в FoxPro должны находиться в диапазоне от 1 января 100 года до 12 декабря 9999 года. Работа с полями типа дата удобна тем, что для них имеется множество специальных функций обработки дат. Ввод дат пользователем сопровождается автоматическим контролем, не допускающим бессмысленные или неразрешенные значения.

Поля дат и логические поля имеют стандартную длину, указанную выше, длины остальных полей определяются пользователем. Кроме того, база данных может быть расширена за счет так называемых полей примечаний (мемо-полей). Эти поля (тип M) имеют произвольную длину в каждой записи. Их содержимым могут быть данные любого вида, включая даже COM- и EXE-файлы. Поля примечаний удобно использовать для такой информации, которая имеет непредсказуемую длину. Например, если в базу данных ввести раз-

дел "Поощрения", ясно, что его длина может быть самой разной, в том числе и равной нулю для лиц, недавно поступивших на работу. В этом случае неразумно отводить поле фиксированной длины в самой базе, а лучше воспользоваться полями примечаний. Кроме того, символьные поля все равно не могут иметь длину более 254 символов, а поля примечаний — практически любую.

Хотя поля примечаний создаются одновременно и одинаковым образом с остальными полями базы данных, хранятся эти поля не в файле DBF, а в специальном, связанном с ним файле примечаний. Файл примечаний имеет одинаковое с файлом базы данных имя, но расширение FPT. В каждой записи файла DBF имеется только ссылка фиксированной длины 10 на каждое имеющееся в базе мемо-поле.

Файлы примечаний являются подчиненными по отношению к файлам DBF. Доступ к полям примечаний становится возможным только в случае, если открыт соответствующий файл DBF.

Временные переменные

Временные переменные (далее просто переменные) создаются и используются в программах и могут быть при необходимости сохранены на диске в файле типа MEM. Ограничения на переменные: максимальная длина одной переменной — 64 Кбайта/2 Гбайта; максимальное число переменных — 3600/65000. В знаменателе указаны значения для расширенной версии FoxPro.

Новым в версии 2.0 является отсутствие "строчного" пула — суммарный размер символьных переменных ограничен только размером свободной памяти. По умолчанию допустимо использование 256 переменных. Чтобы установить иные умолчания, необходимо изменить параметр MVCONT в файле CONFIG.FP FoxPro.

Числовые переменные могут иметь представление как с фиксированной, так и с плавающей точкой. Представление с плавающей точкой применяется в основном для изображения очень больших или очень маленьких чисел. В этом случае число изображается в виде значащей части и степени числа 10 (буква E). Например: 123E+9 равно 123000000000, а 2.6E-7 равно 0.00000026.

В FoxPro допускается определение и использование одномерных и двумерных массивов переменных. Причем каждый элемент массива может быть любого из имеющихся типов независимо от остальных элементов. Максимальное число элементов в массивах 3600.

Имена полей и переменных могут иметь длину до 10 символов (латинских букв, цифр и знаков подчеркивания) и должны начинаться с буквы. Прописные и строчные буквы в именах воспринимаются FoxPro одинаково.

Работа с данными в среде FoxPro

Обработка данных в FoxPro может выполняться с помощью следующих подходов:

- Непосредственная обработка данных пользователем через системные меню FoxPro. Работа на этом уровне требует от пользователя хорошего владения интерфейсом системы и умения читать англоязычные сообщения. Здесь могут быть поставлены лишь очень простые задачи.

- Обработка данных с помощью прикладных программ. Создание программных продуктов в среде FoxPro может быть выполнено только квалифицированным программистом. Использование готовых прикладных пакетов совершенно освобождает пользователя от необходимости что-нибудь

изучать, кроме самой прикладной системы, а также от выполнения каких-либо технических действий по обработке данных и позволяет сосредоточиться только на принятии содержательных решений.

о Обработка данных с помощью программ, созданных средствами генератора приложений.

СУБД FoxPro имеет развитый аппарат создания заготовок программ, которые могут быть применены и адаптированы пользователем для своих целей. Такого пользователя можно условно назвать "квалифицированным" пользователем. От него требуется не только очень хорошее знание интерфейса СУБД, но и некоторые навыки программирования. Генераторы экранов, отчетов и т.п. прежде всего полезны программистам для сокращения рутинных работ.

На практике отечественный пользователь может успешно применять только готовые программные продукты. Это связано с невысокой еще технической культурой управленческого персонала всех уровней и трудностями адаптации к англоязычным пакетам.

Создание программных продуктов

FoxPro обладает эффективным языком программирования прикладных информационных систем. Отдаленно он напоминает такие языки, как Паскаль, ПЛ/1, однако является языком более высокого уровня.

Ввиду того что языки создания информационных систем ориентированы на непосредственный диалог с пользователем по обработке данных, находящихся в дисковых файлах, для них обычно характерны следующие особенности:

- наличие мощных команд обработки файлов;
- развитые средства ведения диалога (меню, "горячие" клавиши);
- удобные средства ввода/редактирования данных;
- возможность ускоренного доступа к данным (индексирование);
- управление дизайном экрана (окна, цвет, звук, рамки);
- удобный вывод данных на экран, бумагу, в текстовый файл;
- развитый аппарат обработки символьных данных.

Всеми этими возможностями в высокой степени обладает язык FoxPro. В то же время здесь, например, отсутствуют средства обработки двоичных и шестнадцатеричных данных, отсутствуют иные агрегаты данных, кроме массивов, маловато математических функций, т.е. нет некоторых популярных языковых средств. Однако при создании систем обработки данных без этого можно обойтись, так как собственно математические вычисления обычно довольно просты, а остальные задачи решаются другими средствами.

Замечание. При чтении материала, вероятно, вызовет недоумение тот факт, что некоторые различные команды и функции выполняют сходные действия. Например, здесь поддерживаются две альтернативные концепции организации меню. Это обстоятельство вызвано конкурентной борьбой между фирмой Fox SoftWare (СУБД FoxPro) и фирмой Ashton-Tate (dBASEIV) за рынок сбыта. Чтобы привлечь к себе пользователей СУБД dBASEIV, в FoxPro были включены практически все команды и функции dBASEIV версии 1.0, даже если подобные средства здесь уже имелись. Ввиду этого команды-дубликаты можно игнорировать.

Глава 2. ОБОЗНАЧЕНИЯ И СТРУКТУРА КОМАНД СУБД

При изучении языка будем использовать для описания синтаксиса команд следующие обозначения:

Файл	— имя файла. Если нужно подчеркнуть тип файла, то может быть указано и расширение его имени (например, DBF-файл для файла базы данных).
БД	— база данных (DBF-файл, возможно, с FPT-файлом).
Поле	— имя поля файла базы данных.
Мемо-поле	— имя поля примечаний из файла примечаний (FPT-файла).
Индекс	— имя индекса (имя индексного IDX/CDX-файла или тэга в CDX-файле (по контексту)).
Перем	— имя временной переменной, находящейся в памяти.
Область	— имя рабочей области, которую организует FoxPro для обработки одного файла базы данных. Если не оговорено иначе, область может быть указана номером, буквой или псевдонимом базы данных.
Окно	— имя окна.
[...]	— в квадратных скобках указывается необязательная, но возможная часть конструкции команды. Скобки в команду не входят.
<...>	— в угловых скобках помещается всякое разрешенное выражение, которое программист должен поместить в команду. Скобки в команду не входят.
.../...	— указывает на то, что в команде необходимо наличие только одного из элементов, разделенных знаком "/".
* и ?	— эти стандартные для MS DOS символы в имени файла и в ключе поиска некоторых команд означают любое количество произвольных символов (*) и один произвольный символ (?). Совокупность указанных символов замещения и известных символов имени образует понятие "маска".
ВырN	— выражение числового типа (Numeric). Его результатом является число.
ВырL	— выражение логического типа (Logical). Вырабатывает значение "Истина" или "Ложь". Логические выражения в синтаксисе команд будем также иногда обозначать термином "условие".
ВырC	— выражение символьного типа (Character). Вырабатывает строку символов или отдельный символ.
ВырD	— выражение типа дата (Date). Его результатом является число, к которому применима арифметика дат.
Выр	— выражение любого типа вообще или любого типа из разрешенных по контексту.

Здесь под "выражением" понимаются все возможные (если не оговорено иначе или не вытекает из контекста) носители информации данного типа в командах: базы данных, константы, переменные, поля записей и их функции. В некоторых сложных случаях в синтаксисе команды в качестве аргумента может использоваться не слово <выр>, а содержательное его значение, например <номер позиции>. Тогда тип соответствующего выражения ясен из контекста (здесь это, конечно, числовой тип, т.е. <вырN>).

В тексте также будет часто встречаться значок "■", с помощью которого выделяется обычно первое описание формата команд и функций.

Команда может иметь длину до 2048 символов. Для переноса в тексте

программы части команды на следующую строку в конце текущей строки ставится знак ";".

Знаки операций

Команды могут содержать следующие знаки операций:

МАТЕМАТИЧЕСКИЕ (перечислены в порядке их приоритетов):

1. ** или ^ — возведение в степень;
2. * — умножение, / — деление, % — остаток от деления;
3. + — сложение, - — вычитание.

ЛОГИЧЕСКИЕ (в порядке приоритетов):

1. NOT — НЕ (другая форма указания операции НЕ — !);
2. AND — логическое И;
3. OR — ИЛИ.

Знаки логических операций (кроме !) окаймляются точками или пробелами.

ОТНОШЕНИЯ: < — меньше, > — больше, = — равно,
— не равно, <= — не больше, >= — не меньше.

Знаки отношения применимы как к числовым выражениям, так и к датам и символьным выражениям. В последнем случае сравниваются, естественно, не сами символы, а их коды. Если сравниваются символьные строки разной длины, сравнение выполняется по длине второго выражения, т.е. выражения, стоящего справа от знака отношения. Остаток левого операнда, превышающий длину правого операнда, игнорируется.

Таким образом, считаются истинными (.Т.) отношения вида

'ПЕТРОВ А.'='ПЕТРОВ А.' и 'ПЕТРОВ А.'='ПЕТ',

но ложными (.F.)

'ПЕТ'='ПЕТРОВ А.' и 'ПЕТРОВ А.'='ПЕТ'.

Эта чрезвычайно важная особенность дает при необходимости возможность организации поиска данных и по неполному ключу, например фамилии по первой (ым) букве(ам).

Тогда более короткое выражение должно стоять справа от знака "=" и из него обычно должны быть удалены концевые пробелы.

Кроме того, есть операции отношения только для символьных выражений:

- \$ — сравнение символьных строк. Операция A\$B даст значение логическая "Истина" (.Т.), если A идентично B, либо A входит в B, и "Ложь" (.F.) в противном случае.
- = — сравнение на полное тождество символьных строк и по длине, и по содержанию, включая пробелы.

СЦЕПЛЕНИЯ: + соединение двух или более строк в одну;
- то же, но пробелы в конце строки, предшествующей знаку "-", помещаются в конец итоговой строки.

В выражениях все операции выполняются слева направо с учетом их приоритетов. Здесь могут широко использоваться круглые скобки.

Состав (синтаксис) команд включает некоторые закрепленные за ними слова (названия команд, функций и т.п.), которые нежелательно применять в других целях, например в качестве имен. Такие слова называются ключевыми, и их можно записывать сокращенно — до четырех символов. На практике, конечно, следует придерживаться сокращенной формы, однако в

учебных целях мы будем стараться указывать ключевые слова целиком.

Исключение составляют ключевые слова NOSHADOW и NOSHOW для которых, как видим, следует указывать не менее пяти символов. Кроме того, ключевые слова вводятся полностью в командах, где на их месте могут быть указаны также имена объектов. Например, возможна такая фраза: IN <имя окна>/SCREEN. Здесь после слова IN может стоять как указание на экран (SCREEN), так и имя окна. Если задать слово SCREEN не полностью (например, SCRE), то FoxPro сочтет его именем окна.

Далее при описании команд будет широко использоваться термин "по умолчанию". Это означает, что, если какие-то элементы команды опущены, им все равно соответствуют некоторые подразумеваемые действия, которые и считаются действиями по умолчанию.

Структура команды СУБД

Команды FoxPro, ориентированные на обработку файлов базы данных, в самом общем виде имеют следующий синтаксис:

- НАЗВАНИЕ [<границы>] [<список выражений>]
[FOR <условие>] [WHILE <условие>]—

Здесь:

- НАЗВАНИЕ — имя команды;
<границы> — границы действия команды, которые могут иметь одно из следующих значений:
ALL — все записи базы данных;
REST — все записи, начиная с текущей, до конца базы;
NEXT <N> — следующие N записей, начиная с текущей;
RECORD <N> — запись номер N;
FOR <условие> — выполнение команды только для записей, отвечающих <условию>;
WHILE <условие> — выполнение команды только до тех пор, пока не перестанет выполняться <условие>.

Слова FOR и WHILE могут присутствовать в команде одновременно. В этом случае WHILE-условие, очевидно, имеет приоритет перед FOR-условием.

П р и м е р:

LIST REST FIELDS fam,tab,dtr FOR fam='П'

Название	Границы	Выражение	FOR-условие
----------	---------	-----------	-------------

Порядок следования элементов команды (за исключением "названия", которое всегда на первом месте) произвольный. По умолчанию, если отсутствуют <границы> или <условия>, сферой действия команды является одна только текущая запись или, наоборот, вся база данных.

Поясним применение команд с условиями. FOR-условие обеспечивает выполнение команды для всех записей файла базы данных или внутри границ, если они указаны. WHILE-условие означает выполнение команды только до тех пор, пока условие истинно. При встрече первой же записи, в которой оно не удовлетворяется, выполнение команды прекращается, несмотря на то, что нижняя граница еще не достигнута и далее, возможно, имеются записи с нужными свойствами.

Поэтому область применения WHILE-условия — это файлы, упорядоченные (физически или индексированием) по полю, в котором анализируется условие. Если ранее каким-либо образом была найдена первая запись, удовлетво-

ряющая условию, то все остальные такие записи находятся ниже рядом. В этом случае использование команд с WHILE-условием предпочтительнее, так как по достижении последней нужной записи выполнение команды прекращается, а в случае FOR-условия поиск записей был бы бессмысленно продолжен до достижения нижней границы действия команды.

Все вышесказанное по отношению к командам, содержащим FOR-условие, справедливо в случае, если не используется оптимизирующая технология отбора данных (технология Rushmore).

Поскольку в некоторых случаях ее применение может повлечь нежелательные результаты, все такие команды допускают включение опции подавления оптимизации — NOOPTIMIZE.

Технология Rushmore будет подробно рассмотрена позже. Сейчас мы упоминаем об этой опции лишь для того, чтобы не останавливаться на ней каждый раз. Далее в некоторых командах с FOR-условием она даже не указана, хотя и присутствует в их формате.

Кроме того, в FoxPro имеется большая группа команд, которые целесообразно выделить в отдельный вид:

- SET <параметр команды> TO <значение параметра>

- SET <параметр команды> OFF/ON

Такие команды (команды-установки), как правило, не влекут каких-то немедленных действий, а определяют условия работы других команд, т.е. останавливают операционную среду FoxPro. Параметр может быть задан некоторым <значением> или включен/выключен (ON/OFF).

В FoxPro разрешается и ключевые слова, и имена записывать как прописными, так и строчными буквами. В дальнейшем в примерах мы будем ключевые слова команды записывать прописными буквами, а остальное — строчными.

Это, конечно, не относится к данным. Если предполагается поиск в базе, то ключ поиска должен в точности совпадать с искомыми элементами данных. Совпадение должно быть полным, включая прописные и строчные буквы, поскольку они имеют разные коды представления в компьютере.

Константы различных типов данных отображаются в командах следующим образом:

Символьные константы в FoxPro выделяются апострофами, кавычками или квадратными скобками. В случае, если сами символы-ограничители являются элементами данных, вся строка должна быть заключена в другие разрешенные ограничители, например [Кинотеатр "Ударник"]. Мы будем пользоваться апострофами, например 'Петров А.'.

Константы типа дата берутся в фигурные скобки (например, {09.11.91}).

Пустая дата ({ . . }) вообще может быть задана одними фигурными скобками {}.

Логические константы Т и F в тексте программ обрамляются точками (.Т., .F., .t., .f.).

Далее в книге приводятся не все команды, а только основные, и в них в некоторых случаях рассматривается не полный синтаксис, а его важная часть.

Сейчас рассмотрим, а если возможно, и опробуем в интерактивном режиме (в окне Command) основные команды СУБД.

Глава 3. СОЗДАНИЕ ФАЙЛА БАЗЫ ДАННЫХ

Создание файла базы данных включает два этапа: создание структуры файла и его заполнение данными.

3.1. Создание структуры файла

Действие команды рассмотрим на примере. Создадим простой файл базы данных, который содержит сведения о кадровом составе предприятия, включающие следующие данные (названия полей указаны в скобках):

1. Фамилия и инициалы работника (FAM);
2. Дата рождения (DTR);
3. Табельный номер (TAB);
4. Количество детей (DET);
5. Пол (POL);
6. Семейное положение (SEM);
7. Средняя зарплата (SZAR);
8. Подразделение, место работы — отдел, цех (PODR);
9. Сведения о перемещениях по службе (PER).

В поле ПЕРЕМЕЩЕНИЯ (PER) будем заносить информацию о местах работы и должностях сотрудника внутри предприятия.

Назовем файл базы данных KADR.DBF. Выберем для его полей типы и размеры:

1. FAM — символьный тип (Character) длиной 25 символов;
2. DTR — тип дата (Date) со стандартной длиной 8;
3. TAB — числовой (Numeric) тип длиной 3 разряда целых;
4. DET — числовой тип длиной 1 разряд целых;
5. POL — символьный тип длиной 1 символ (М или Ж);
6. SEM — символьный тип длиной 1 символ (допускаются значения: Б — в браке, Х — холост, Р — разведен);
7. SZAR — числовой тип общей длиной 7 разрядов, из которых два разряда дробных (максимальная зарплата 9999.99 руб.);
8. PODR — символьный тип длиной 15;
9. PER — ввиду непредсказуемости длины этой информации возьмем для нее поле примечаний (тип Метод-поле).

Структура файла базы данных типа DBF создается командой

■ CREATE <имя файла>

В нашем случае для создания файла KADR.DBF — это команда

```
CREATE kadr
```

Точка перед командой указывает на непосредственный ввод ее в командном окне. Сама точка не вводится.

Расширение имени файла DBF указывать необязательно, так как оно добавляется автоматически. Если файл создается не на активном в данный момент диске и/или директории, нужно указывать и дисковод, и путь доступа, например диск B

```
.CREATE b:kadr
```

В ответ СУБД представит экран-форму для ввода данных о структуре создаваемого файла базы данных, а именно для каждого вводимого поля — его имя, тип, длину и для числового поля — точность (число дробных позиций).

В нашем случае для файла KADR.DBF заполним предлагаемую форму, как показано на рис.3.1.

Тип поля выбирается в момент, когда курсор находится в ко-лонке TYPE, нажатием первой буквы указания типа (C, N, F, D, L, M) или клавиши Space/Enter (Пробел/Ввод). В ответ появляется меню выбора (рис.3.2). В нижней части меню определения структуры ведется статистика. Из рис.3.1 видно, что база содержит 9 полей общей длиной 72 байта. Еще доступно 3928 байт (из максимально возможных 4000).

Порядок расположения полей может быть изменен. Возможно удаление имеющихся и дополнение новых полей (<Insert> и <Delete>).

Structure: D:\FOXPRO\KADR.DBF

Name	Type	Width	Dec
FAM	Character	25	
DTR	Date	8	
TAB	Numeric	3	0
DET	Numeric	1	0
POL	Character	1	
SEM	Character	1	
SZAR	Numeric	7	2
PODR	Character	15	
PER	Memo	10	

Fields: 9 Length: 72 Available: 3928

Рис.3.1

Field

<Insert>

<Delete>

< OK >

<Cancel>

Character

Numeric

Float

Date

Logical

Memo

Picture

Рис.3.2

Замечание. Тип Picture имеет смысл только для ПК "Macintosh" и не может быть назначен на IBM PC.

После окончания формирования структуры файла она должна быть запомнена на диске. Это осуществляется одновременным нажатием клавиш Ctrl-End или Ctrl-W. Нажатие клавиши Escape вызовет отказ от сохранения структуры. Тот же результат может быть достигнут нажатием Enter после перемещения курсора в позицию <OK> или <Cancel> соответственно. Все действия по управлению созданием базы могут быть реализованы и мышью.

Затем FoxPro запросит ввод данных:

Input data records now?

< Yes > < No >

Ответим No.

Если в дальнейшем обнаружится, что структура базы данных нас не удовлетворяет, ее можно изменить командой модификации структуры

■ MODIFY STRUCTURE

При этом мы попадем в меню, идентичное меню соманды CREATE, где и увидим структуру модифицируемого файла. Здесь можно удалять, переименовывать и дополнять поля в базе данных, а также изменять их параметры. Если в файле базы данных к этому моменту имелись данные, они будут (если это возможно) сохранены. Модифицируемый файл должен быть предварительно открыт. При модификации базы данных старые структуры сохраняются на диске с расширениями имен ВАК для DBF-файлов и ТВК для FPT-файлов.

Прежде чем перейти к следующему материалу командой

■ SET STATUS ON

установим на экране статус-строку (для удобства ориентирования в среде FoxPro). В этой строке будет содержаться полезная для пользователя, а иногда и для программиста информация: имя выполняемой программы (если

есть), активный диск, имя открытой базы, номер текущей записи, общее число записей в базе данных (эти параметры отображаются в виде дроби как числитель и знаменатель), признак пометки текущей записи к удалению (слово Del), положение клавиш Num Lock и Caps Lock (Num и Caps). Удаляется статус-строка командой SET STATUS OFF (эта форма действует по умолчанию). Статус-строка полезна при обучении и отладке, но в готовых программах она, конечно, не нужна.

Далее при работе в командном окне мы увидим, что результаты выполнения многих команд автоматически отображаются на экране. Такое качество FoxPro определяется командой

■ SET TALK ON/OFF

По умолчанию ON. Программисту эта особенность обычно только мешает, и она должна быть подавлена (SET TALK OFF).

3.2. Заполнение базы данных

Открытие файла базы данных

Файл после создания структуры остается открытым, т.е. доступным для команд ввода, просмотра и изменения. Однако, если СУБД только загружена в память, должно быть выполнено открытие нужного файла базы данных командой открытия

■ USE [<DBF-файл>]

Команда USE без имени файла закрывает базу данных. Более полный синтаксис команды мы рассмотрим позже.

Закрытие всех файлов баз данных и связанных с ними файлов других типов индексов, форматов, отчетов и т.д. во всех рабочих областях с переходом в первую рабочую область осуществляется командой

■ CLOSE DATABASE

Закрытие вообще всех файлов выполняется командой

■ CLOSE ALL

Дополнение базы данных

Дополнение файла новыми записями осуществляется командой

■ APPEND [BLANK]

которая предьявляет окно ввода данных со всеми пустыми полями создаваемой записи с выделенными другим цветом областями ввода. При этом все поля имеют значения пробелов. Необязательная фраза BLANK означает, что новая запись останется пустой и не будет отражена на экране.

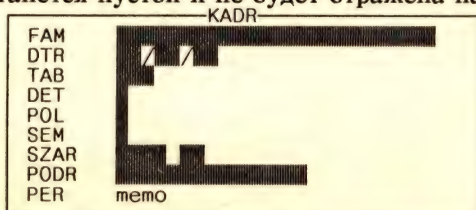


Рис.3.3

В нашем случае команды

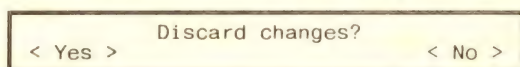

```
.USE kadr
.APPEND
```

обеспечат доступ к окну редактирования данных (рис.3.3) к самой первой записи незаполненного пока файла.

Все поля базы выделяются контрастным цветом в соответствии с их длиной. Мемо-поле указывается только словом "memo".

Заполним новую запись по своему усмотрению. По умолчанию в системе используется американский формат даты (соответствующие команды см. ниже).

После ввода текущей записи автоматически появляется доступ к следующей записи и т.д. Для того чтобы войти в мемо-поле, необходимо в него поместить курсор и нажать Ctrl-Home. Выход из мемо-поля с сохранением сделанных в нем изменений на диске осуществляется клавишами Ctrl-W или Ctrl-End, без сохранения — Escape. При этом, если изменения в мемо-поле были сделаны, FoxPro выдает вопрос-предупреждение "Не сохранять изменения ?":



Ответ Yes завершит работу в окне, No вернет нас обратно в окно ввода.

Выход из всего окна APPEND осуществляется аналогичным образом. При нажатии Escape не сохраняется только новая информация в поле, в котором находится курсор.

В окне редактирования возможны возврат к предыдущим записям для их изменения и вообще листание записей (клавиши PgUp/PgDn). Допускается также пометка записей к удалению нажатием Ctrl-T. Более подробно клавиши управления редактированием будут рассмотрены позже.

По умолчанию в FoxPro принят американский (AMERICAN) формат даты — две цифры месяца, дня и года, отделенные косой чертой, т.е. ММ/ДД/ГГ. Очевидно, что для нас это очень неудобно.

Имеется возможность установить иные формы даты командой

■ SET DATE <тип даты>

Приведем ее важнейшие типы и форматы:

SET DATE AMERICAN	— ММ/ДД/ГГ;
SET DATE ANSI	— ГГ.ММ.ДД;
SET DATE BRITISH/FRENCH	— ДД/ММ/ГГ;
SET DATE GERMAN	— ДД.ММ.ГГ;
SET DATE ITALIAN	— ДД-ММ-ГГ.

Кроме того, можно при необходимости отображать дату не двумя, а четырьмя цифрами года (параметр — ON) с помощью команды

■ SET CENTURY OFF/ON

По умолчанию OFF. Остановимся на типе, определяемом командой

```
.SET DATE GERMAN
```

которая установит привычный для нас порядок следования элементов даты — ДД.ММ.ГГ. Так, 25 марта 1946 г. будет записано как 25.03.46. Чтобы установить этот формат даты, введите команду в окне Command. В программах ее следует поставить в начало командного файла или включить в файл конфигурации CONFIG. FP.

В дальнейшем будем вводить и изображать дату именно таким образом, имея в виду, что указанная команда уже выполнена.

По умолчанию ввод/редактирование данных в команде APPEND и во всех

других командах редактирования имеет две особенности. При вводе символа в последнюю позицию очередного поля раздается звуковой сигнал и курсор автоматически (без нажатия клавиши Enter) переходит на первую позицию следующего поля данных, что облегчает ввод данных вслепую, если они имеют фиксированную длину. Этим процессом можно управлять с помощью команд

■ SET BELL ON/OFF

— включение/отключение звукового сигнала;

■ SET CONFIRM ON/OFF

— включение/отключение автоматического перехода курсора на следующее поле.

По умолчанию команды имеют установки ON. Если ввод вслепую неудобен, опасен или вас раздражают эти эффекты, их можно подавить (OFF). Кроме того, команда SET CONFIRM влияет на выбор в меню с "горячими" клавишами.

При отладке и/или работе прикладной системы может возникнуть необходимость обращаться в несколько директорий. Эту возможность предоставляет команда

■ SET DEFAULT TO [<путь>]

которая устанавливает диск и/или директорий в качестве используемого по умолчанию. По этой команде выполняется команда CD (смена директории) операционной системы. Далее розыск имеющихся файлов и создание новых файлов будут выполняться именно здесь. В начале директорией по умолчанию считается стартовая директория, откуда был выполнен вызов FoxPro.

Пр и м е р. Задание активного диска/директории D:\PLAN:

```
.SET DEFAULT TO D:\PLAN
```

Команда

■ SET PATH TO [<список путей>]

задает пути (маршруты) только поиска файлов, не обнаруженных в текущей директории. Текущими остаются диск/директория, назначенные командой SET DEFAULT TO. Кроме собственных маршрутов FoxPro может "обследовать" и маршруты DOS. Выяснить имя текущей директории, а также пути доступа к нужным файлам можно с помощью функций SYS(5), SYS(2003), SYS(2004) и др. (см. гл.16).

Глава 4. ОКНО РЕДАКТИРОВАНИЯ

При выдаче команд APPEND, INSERT, EDIT, CHANGE, BROWSE и наличии открытой базы данных FoxPro разворачивает для пользователя окно редактирования. Вся информация в окне доступна для изменения. Кроме того, возможны дополнение базы и удаление записей.

Средства, предоставляемые в окне редактирования, не исчерпываются обзором и редактированием данных. Здесь может быть организован очень удобный интерфейс обработки данных, включающий входной контроль данных, необходимые заголовки и предупреждения, управление цветами, вычисляемые поля и т.д.

Стандартное окно редактирования имеет две формы. Для первых четырех команд оно будет выглядеть одинаково — все поля базы данных располагаются вертикально (см. выше экран к команде APPEND). На экране видно столько записей и полей, сколько удастся их здесь разместить. Назовем форму такого окна CHANGE-окном.

Другую форму предъявления данных осуществляет команда BROWSE (BROWSE-окно). Здесь все поля каждой записи располагаются горизонтально — колонками. Если какие-то поля записи не умещаются в строке, с помощью клавиш управления курсором и мышью возможно перемещение (скролинг) изображения вправо или влево.

Выбор формы CHANGE или BROWSE представления данных определяется структурой базы данных и удобством пользователя. Базы, содержащие большое число полей, удобнее представлять в формате CHANGE. Тогда весь экран может быть отведен под изображение одной записи. Вместе с тем на одном экране возможно отображение данных сразу в обоих форматах.

Клавиши управления действуют практически одинаково в обеих формах окон. Ниже они описаны. Пометка записей к удалению выполняется клавишами Ctrl-T. Признаком пометки является появление специального значка (точки) в самой левой колонке записи. Это же действие может быть реализовано с помощью мыши, для чего ее маркер должен быть установлен в самую левую колонку и нажата кнопка мыши.

Дополнение базы новой записью осуществляется нажатием клавиш Ctrl-N (кроме команды APPEND, в которой это происходит автоматически). Можно установить режим копирования в новую запись всех (ON) полей из текущей (не обязательно последней) записи командой

■ SET CARRY ON/OFF

По умолчанию OFF. Можно установить и выборочное копирование только указанных <полей> командой

■ SET CARRY TO [<поля>]

Если параметр <поля> опущен, происходит возврат к действию команды SET CARRY ON/OFF.

Режим копирования может быть очень полезен при вводе данных с повторяющимися значениями полей. Копирование не происходит при добавлении новых записей иным образом, например командой APPEND BLANK.

Перемещение внутри базы данных осуществляется с помощью клавиш перемещения курсора или мыши. Кроме того, могут использоваться следующие клавиши:

- | | |
|---------------|--|
| Home/End | — переход к началу/концу поля; |
| Tab/Shift-Tab | — переход к следующему/предыдущему полю; |
| PgUp/PgDn | — переход назад/вперед на один экран; |
| Enter | — переход к следующему полю; |

Ctrl-Y	— удаление поля;
Ctrl-Home, Ctrl-PgUp, Ctrl-PgDn или двойное нажатие кнопки мыши	— вход в мемо-поле;
Ctrl-W/End	— выход с сохранением измененных данных;
Escape	— выход без сохранения.

В последнем случае не сохраняются только изменения, сделанные в текущем поле, если курсор не был из него выведен. Изменения, сделанные в других полях, сохраняются.

Над полями базы данных можно осуществлять действия, предусмотренные для встроенного редактора FoxPro: выделение фрагментов данных, их копирование и удаление (см. описание редактора FoxPro).

Для предъявления CHANGE-окна введите команду CHANGE. При этом вы увидите уже знакомую по команде APPEND форму отображения данных.

При вводе команды BROWSE появляется изображение данных, показанное на рис.4.1. Форма окна приведена с некоторыми упрощениями.

Fam	Dtr	Tab	KADR		Det	Podr	Szar	Per
			Pol	Sem				
СИДОРОВ П.С.	12.10.56	13	М	Х	1	ОГМ	1350.00	memo
ПОТАПОВ Д.П.	04.09.60	98	М	Б	3	ОГМ	1280.00	memo
КУЛАКОВА М.И.	15.04.49	6	Ж	Б	2	ОГМ	900.00	memo
ПОПОВ А.А.	25.03.46	234	М	Р		КБ	1500.00	memo
РОМАНОВА М.С.	09.10.66	890	Ж	Х		ОК	950.00	memo
МИРОНОВ Р.И.	09.09.70	468	М	Х		ОГМ	1420.00	memo
ЯКОВЛЕВ А.И.	10.12.30	54	М	Б	2	ВОХР	2180.00	memo

Рис.4.1.

В мемо-поле стоит слово "мемо". Если оно начинается со строчной буквы, то поле пустое, если с прописной ("Мемо"), то в нем имеются данные. Чтобы войти в мемо-поле, необходимо переместить в него курсор и нажать Ctrl-Home или Ctrl-PgDn/PgUp.

По умолчанию окно редактирования может быть видно не целиком. Раскрывается окно во весь экран нажатием клавиш Ctrl-F10. Повторное нажатие клавиш возвращает окно к исходной форме. В дальнейшем мы рассмотрим средства управления размером и положением окон.

Недостатком стандартных форм как BROWSE-окон, так и CHANGE-окон является их неудобство для русскоязычного пользователя. Естественно, хотелось бы видеть названия полей написанными не латинскими, а русскими буквами. Кроме того, при вводе желательно предусмотреть контроль данных. Команды BROWSE/CHANGE позволяют сделать это и многое другое. Изучим их подробнее.

Здесь могут использоваться разнообразные (и еще не рассмотренные) средства языка (индексирование, функции, шаблоны). Поэтому, хотя опережающий материал по возможности разъяснен, при первом чтении можно его опустить либо заглянуть вперед. Такой же практики будем придерживаться и в дальнейшем.

Замечание. Далее в тексте будет широко применяться понятие "Пользовательская функция" (ПФ). Этот термин означает наличие в программе внешней или внутренней процедуры-функции, к которой можно обратиться по имени и в которой выполняются некоторые действия по обработке данных.

4.1. BROWSE-окно

Команда BROWSE — один из наиболее мощных и удобных инструментов доступа пользователя к данным в FoxPro. По существу это не просто команда — это целая среда доступа и управления данными.

Записи из базы (по умолчанию) предъявляются горизонтально на экране или внутри предварительно описанного окна. Записи можно редактировать, дополнять и помечать к удалению. Формат отображения полей в BROWSE-окне может настраиваться пользователем.

Допускается создавать так называемые вычисляемые поля. Эти поля фактически не являются полями базы данных, но могут быть их функциями и отображаются на экране наравне с настоящими полями, что дает возможность пользователю, например, оценивать свои данные по заданному критерию. Такой режим соответствует работе с электронной таблицей.

Если позволяет основная память компьютера, то можно одновременно открыть до 25 BROWSE-окон.

Команда BROWSE дает возможность предъявлять поля из разных баз данных. Кроме того, она поддерживает любой тип связи между базами (см. гл.10). При выполнении команды можно получить текущее значение колонки/строки относительно окна/экрана (функции COL()/ROW()), имя текущего поля (функции VARREAD() и SYS(18)), номер текущей записи (RECNO()) и т.д. Допускается широкое применение аппарата пользовательских функций. Формат команды:

■ BROWSE

[FIELDS <поля>]	[FOR <условие1>]
[FORMAT]	[FREEZE <поле>]
[KEY <выр1>[,<выр2>]]	[LAST]
[LEDIT/REDIT]	[LOCK <вырN1>]
[LPARTITION]	[NOAPPEND]
[NOCLEAR]	[NODELETE]
[NOEDIT/NOMODIFY]	[NOLGRID/NORGRID]
[NOLINK]	[NOMENU]
[NOOPTIMIZE]	[NORMAL]
[NOWAIT]	[PARTITION <вырN2>]
[REST]	[PREFERENCE <вырC1>]
[TIMEOUT <вырN3>]	[TITLE <вырC2>]
[NOOPTIMIZE]	[WHEN <условие3>]
[WINDOW <окно>]	[NOWAIT]
[VALID [F:] <условие2> [ERROR <вырC3>]]	
[COLOR SCHEME <вырN5>/COLOR <список цветовых пар>]	

Рассмотрим действие опций команды, разбив их на группы.

Управление доступом к полям базы

FIELDS <список полей> — перечень предъявляемых полей. По умолчанию отображаются все поля базы данных. Имя каждого поля может сопровождаться ключами, определяющими режим доступа к нему:

```
[ :R]
[ :<вырN>]
[ :V = <вырL1> [ :F] [ :E = <вырC1>]]
[ :P = <вырC2>]
```



```
[ :H = <вырС3>]
[ :B = <выр1>, <выр2> [ :F] ]
[ :W = <вырL2>]
```

Здесь символ ":" может быть заменен на тождественный ему символ "/", если последний не будет интерпретироваться как знак деления (это возможно для параметра :<вырN>).

Перечисленные ключи имеют следующие значения:

- :R — разрешен только просмотр поля (Read-Only), однако курсор в поле допускается и, следовательно, в нем, например, могут быть обработаны нажатия заданных клавиш, а также задействованы ключи V и W.
- :<вырN> — видимый размер поля в BROWSE-окне. Если фактический размер поля больше, допускается скролинг.
- :V=<вырL1> — контроль выхода из поля. Проверка вводимых данных выполняется по <вырL1> после изменения содержимого поля. Если <вырL1>=.T., ввод считается правильным и курсор переходит в следующее поле. Если <вырL1>=.F., выдается стандартное сообщение "Invalid input" ("Неверный ввод"), которое может быть заменено на собственное, заданное с параметром :E. Допустимы ПФ. Ключ V соответствует опции VALID команды ввода @ ... GET, которая будет рассмотрена позже.
- :F — проверка не только вводимых, но и уже существующих данных независимо от способа выхода из поля.
- :E=<вырC1> — выдача собственного <вырC1> сообщения на неправильный ввод данных. Если <вырL1> вычисляется как числовое и в результате получено значение 0, сообщение не выдается и выхода из поля не происходит. Это дает возможность создавать собственные подпрограммы сообщений об ошибках из процедур обработки ошибок. По действию ключ аналогичен опции ERROR команды @ ... GET.
- :P=<вырC2> — задание формата отображения данных с помощью шаблона Picture или символов форматной функции ввода-вывода (кроме кода "M"). Шаблоны и коды форматных функций описаны в команде @ ... SAY ... GET при рассмотрении опций PICTURE и FUNCTION.
- :B=<выр1>, <выр2> — указание границ чисел или дат. Не допускаются ПФ. Возможно указать только одну из границ, но запятая должна присутствовать обязательно. Может быть усилена параметром :F. Ключ B соответствует опции RANGE команды @ ... GET.
- :H=<вырC3> — указание собственного заголовка поля. Разрешены ПФ. По умолчанию в качестве заголовков выводятся имена полей. Если заголовок не нужен вообще, следует в качестве <вырC3> использовать пробел (' ').
- :W=<вырL2> — контроль входа в поле. Запрещены вход и редактирование, если <вырL2>=.F., и разрешены, если <вырL2>=.T.. Допустимы ПФ. Входом в поле считается перемещение в него курсора любым способом, в том числе и мышью. Ключ W соответствует опции WHEN команды @ ... GET.

Вычисляемые поля

В <список полей> могут включаться вычисляемые поля. Эти поля являются функциями других полей, переменных и т.д. Такие поля не могут редактироваться и не запоминаются в базе данных.

Вычисляемые поля сами могут содержать пользовательские функции, что делает их важным средством отображения и управления данными.

Например, в команде BROWSE для базы KADR.DBF введем вычисляемое поле POM для определения материальной помощи. Считаем, что помощь устанавливается на одного ребенка в размере 70% средней зарплаты, но не более 900 руб. и только тем, у кого средняя зарплата не превышает 1600 руб. Таким образом $POM = IIF(szar > 1600, 0, MIN(0.7 * szar * det, 900))$.

Здесь, забегая вперед, мы использовали две новые функции: MIN() и IIF(). Функция MIN() возвращает минимальное значение из $0.7 * \text{szar} * \text{det}$ и 900. Функция IIF() возвращает 0, если $\text{szar} > 1600$, или MIN(...) в противном случае. Подробнее эти функции разбираются в гл.16.

Видимый размер вычисляемого поля РОН будет определяться принятыми умолчаниями на размер числовых выводов. Этим процессом можно (и лучше) управлять с помощью параметра ограничения длины поля <вырN>. Еще удобнее применение шаблонов (ключ :E), которыми может быть установлена не только длина, но и формат выдачи.

Отбор данных

Следующие исключительно важные опции позволяют ограничить доступ в базе только к определенному множеству записей.

FOR <условие1> — устанавливает фильтр записей для базы. В BROWSE-окне предъявляются только записи, удовлетворяющие заданному <условию>.

Пример:

```
.USE kadr  
.BROWSE FOR szar>=5300.AND.szar<=7500
```

Здесь команда BROWSE предъявляет только те записи базы данных KADR.DBF, в которых значения поля SZAR (средняя зарплата) от 5300 до 7500 руб. Если имеются соответствующие индексные файлы, фильтрация данных будет выполняться с их участием и использованием оптимизирующей технологии Rushmore. Такой индекс, естественно, должен быть открыт. Опция NOOPTIMIZE отключает оптимизацию.

REST — удобно использовать совместно с FOR-условием. Это предотвращает повторный поиск в базе с самого ее начала записей, отвечающих <условию> при повторном вызове BROWSE-окна. Курсор остается на текущей записи, если, конечно, она удовлетворяет <условию>.

KEY <выр1> [, <выр2>] — ограничение действия команды диапазоном ключевого выражения <выр1> и <выр2> активного индексного файла.

Пример:

```
.USE kadr  
.INDEX ON szar TO kadrzar  
.BROWSE KEY 5300, 7500
```

Пример по целям аналогичен предыдущему, но для функционирования такого режима доступа к данным уже обязательно наличие предварительно созданного индексного файла KADRZAR.IDX по полю SZAR (см. команду INDEX).

По возможности следует использовать гораздо более быстрый способ фильтрации с применением индексного файла. Для этого могут служить опция KEY, включение FOR-условия при наличии необходимых открытых индексов (по технологии Rushmore) или открытие базы с индексом, который сам построен с FOR-условием. Индексирование и технология Rushmore будут рассмотрены далее.

Разделение окна

В режиме разделения BROWSE-окно разбивается на два смежных окна, в которых отображаются данные из базы/баз, возможно, в разной форме. Это может быть удобно, например, если надо сделать неподвижной какую-то часть

данных в одном окне при горизонтальном или вертикальном перемещении в другом.

LOCK <вырN1> — BROWSE-окно делится на две части, где первые <вырN1> полей всей базы или из списка FIELDS (если есть) дублируются в левой части окна. Перебрасывается курсор в другую половину окна нажатием клавиш Ctrl-H или мышью.

PARTITION <вырN2> — то же, но граница будет проходить по колонке номер <вырN2>.

LEDIT/REDIT — опции действуют только в разделенном окне. Они указывают, будут ли слева/справа (LeftEDIT/RightEDIT) от линии разделения данные показаны, как в CHANGE-окне (EDIT — другое имя команды CHANGE). В противоположной части окна поля останутся расположенными горизонтально.

LPARTITION — курсор устанавливается в левой половине разделенного окна (по умолчанию — в правой половине).

NOLINK — несинхронное перемещение видимых записей в разделенных окнах. При движении курсора в одной половине окна записи в другой половине остаются неподвижными. Это удобно, например, в случае, если нужно получить доступ сразу к двум записям одной базы данных или если установлена связь между базами типа один-ко-многим для перемещения в подчиненной базе при неподвижной записи из старшей базы. По умолчанию записи синхронно перемещаются в обеих половинах окна.

NOLGRID/NORGRID — удаляет вертикальные линии-разделители полей в левой/правой части разделенного окна. По умолчанию разделители есть. Если окно не разделено, вертикальные линии удаляются опцией NORGRID.

Контроль редактирования записей

Следующие опции аналогичны рассмотренным ранее ключам :V, :W, :E и :F при определении возможности выхода из текущего поля, но распространяются на всю запись целиком. При этом могут анализироваться любые условия:

VALID <условие2> [ERROR <вырC3>] — анализирует выполнение <условия> для текущей записи, если в ней произошло изменение данных и вы хотите перейти к другой записи. Если условие истинно (.T.), запись покинуть разрешается, если ложно (.F.) или равно 0 — нет, и при этом появляется стандартное сообщение об ошибке "Invalid Input" либо любое другое, заданное в <вырC3>. Контроль может быть усилен включением параметра :F. При этом проверка условия будет выполняться независимо от того, меняли вы данные в записи или нет.

WHEN <условие3> — определяет возможность доступа к записи. Если <условие> ложно или был возвращен 0, запись доступна только для чтения.

Все опции и ключи доступа к полям/записям базы данных позволяют не только организовать контроль ввода данных, но реализовать и более широкие функции.

Ограничение возможности редактирования данных

NOAPPEND — дополнение базы с помощью клавиш Ctrl-N невозможно.

NOEDIT/NOMODIFY — редактирование невозможно. Разрешены пометка к удалению и дополнение базы.

NODELETE — пометка к удалению записей базы данных с помощью клавиш Ctrl-T или мышью невозможна.

FREEZE <поле> — указывает единственное <поле>, значение которого разрешается редактировать. Остальные поля только отображаются.

Все перечисленные опции ограничивают непосредственные действия по изменению данных пользователем. Однако такая возможность остается, например, с помощью команд, вызванных через пользовательские функции. Полное блокирование доступа к данным для редактирования может быть осуществлено открытием/переоткрытием базы командной USE с опцией NOUPDATE (см. описание команды USE в гл.9).

Конфигурирование BROWSE-окна

LAST — конфигурация BROWSE-окна сохраняется в специальном системном так называемом ресурсном файле FOXUSER.DBF (если установлено SET RESOURCE ON) по завершении команды BROWSE для использования в следующем сеансе. Если выход был сделан по Ctrl-Q, конфигурация не сохраняется.

PREFERENCE <вырC1> — работает аналогично опции LAST, но дает возможность сохранить под определенным именем <вырC1> параметры BROWSE-окна в файле FOXUSER.DBF для последующего использования. Конфигурация (и не одна), сохраненная с опцией PREFERENCE, может быть "заморожена" в целях дальнейшего использования. Для этого после выхода из команды BROWSE ресурсный файл должен быть отключен, а затем открыт командой USE и вызван на редактирование, например командой BROWSE. В базе FOXUSER.DBF ищется запись, где в поле NAME стоит <вырC1>, а затем в поле READONLY значение .F. заменяется на .T. (изменение запрещено). После этого ресурсный файл закрывается и снова подключается к системе. Теперь каждая загрузка команды BROWSE будет извлекать конфигурацию BROWSE-окна из ресурсного файла, причем все пользовательские настройки, произведенные в текущем сеансе, в следующих сеансах будут игнорироваться. Более того, в ресурсном файле под разными именами можно сохранить несколько разных настроек для одной базы данных, которые можно выбирать, например, через программируемое меню, перед загрузкой команды BROWSE. Команда BROWSE, использующая сохраненные настройки, в дальнейшем может применяться без всяких опций (кроме опции PREFERENCE) вообще. Более подробная информация о ресурсном файле содержится в гл.31.

FORMAT — предполагает использование форматного файла, который описывает форму предъявления данных, включая: список полей; VALID-, WHEN-, RANGE-условия; шаблоны PICTURE; SAY-выражения для вычисляемых полей. Например, подключение форматного файла может быть сделано следующим образом:

```
.USE kadr  
.SET FORMAT TO kadr.fmt  
.BROWSE FORMAT
```

Расширение имени форматного файла указывать необязательно. Если оно опущено, подразумевается расширение FMT. Пример самого форматного файла будет рассмотрен позже.

NOCLEAR — после выхода из BROWSE-окна оно не удаляется и остается на экране, хотя и не является активным. Затем оно может быть стерто командой CLEAR

NOMENU — подавляет вызов системного BROWSE-меню. Такое меню (оно поддерживается и в исполнительной среде FoxPro) может быть вызвано нажатием клавиши F10. Подавление системного меню одновременно исключает возможность пометки записи к удалению (по Ctrl-T) и дополнения базы (Ctrl-N) и используется в случае, если все клавиши должны быть обработаны программно.

TIMEOUT <вырN3> — указывает время (в секундах), в течение которого FoxPro ожидает ввода данных. Если ввод не произошел, BROWSE-окно закрывается.

TITLE <вырC2> — задает заголовок (<вырC2>) BROWSE-окна.

WINDOW <окно> — позволяет открыть BROWSE-окно внутри другого <окна>, которое предварительно определено командой DEFINE WINDOW. Окно может быть и не активным в данный момент. BROWSE-окно при этом принимает все свойства, установленные для этого окна.

NOWAIT — открытие BROWSE-окна не прерывает выполнения программы. Она продолжается с команды, следующей за BROWSE. Этот режим удобен в программах, предполагающих открытие сразу нескольких, возможно, связанных окон BROWSE, областей редактирования GET, меню и других средств управления. При этом, чтобы не произошло завершение программы, такие окна следует "подвесить" командой READ. Этот механизм будет рассмотрен позже.

COLOR SCHEME <вырN5>/COLOR <список цветовых пар> — определяет раскраску текущего BROWSE-окна с помощью указания номера (<вырN5>) цветовой схемы или непосредственно <списком цветовых пар>. По умолчанию используется цветовая схема номер 10 (см. гл.19).

Загрузив BROWSE-окно, можно не только выполнить редактирование и перемещения в базе, но и настроить его облик под конкретный ввод. Во-первых, это можно осуществить через BROWSE-меню, которое хотя и доступно в прикладных программах, но неудобно для отечественного пользователя. Во-вторых, это можно осуществить мышью. Так, если нажать и удерживать кнопку мыши, когда маркер находится на заголовке колонки, можно изменить положение данной колонки среди других ("отбуксировать" колонку). Если нажать кнопку, когда маркер находится на вертикальной разделительной линии, можно изменить видимую ширину колонки и даже вообще ее скрыть. При желании текущая конфигурация окна сохраняется в специальном ресурсном файле FoxPro с помощью опции LAST или PREFERENCE.

П р и м е р. Настроим окно редактирования таким образом, чтобы оно имело собственные заголовки колонок FAM, DTR, POL, DET, SEM, SZAR, заголовок окна — КАДРЫ, указания на возможные действия в окне (Ctrl-T, Ctrl-N, Ctrl-W), ограничение на верхнюю границу средней зарплаты в 9000 руб. и вычисляемое поле POM (Помощь).

```
.USE kadr
.SET DATE GERMAN
.BROWSE:
  TITLE 'T-удал., N-доп.          КАДРЫ          W-выход';
  FIELDS fam :H='Фамилия' :12.;
         dtr :H='Родился';;
         pol :H='Пол';;
         det :H='Детей';;
         sem :H='Сем. пол.';;
         szar :H='Ср. зар.' :B=,9000 ;;
         pom=IFF(szar>1600,0,MIN(det*szar*0.7,900));
              :H='Помощь' :P='###.##' LEDIT
```

Поскольку из-за заголовков колонок ширина большинства полей увеличилась, видимая часть поля FAM здесь уменьшена до 12 колонок с

возможностью прокрутки этого поля с помощью клавиш управления курсором. Кроме того, для вычисляемого поля РОМ принят шаблон "###.##" (три разряда целых и два дробных). Иначе результат в этом поле будет отображаться разрядностью по умолчанию. Вид нашего BROWSE-окна приведен на рис.4.2.

Т-удал.	Н-доп.	КАДРЫ			W-выход	
Фамилия	Родился	Пол	Детей	Сем. пол.	Ср. зар.	Помощь
СИДОРОВ П.С.	12.10.56	М	1	X	1350.00	900.00
ПОТАПОВ Д.П.	04.09.60	М	3	Б	1280.00	896.00

Рис.4.2

Как видим, предъявление данных вполне удовлетворительно. Вместе с тем имеется возможность еще и настройки конфигурации окна редактирования непосредственно пользователем внутри прикладной программы. Для этого удобнее всего использовать мышь. Возможен, например, экран, изображенный на рис.4.3.

Т-удал.	Н-доп.	КАДРЫ				
Фамилия	Сидоров П.С.	Фамилия				
Детей	1	Сидоров П.С.				
Ср. зар.	1350.00	Потапов Д.П.				
Помощь	900.00	Попов А.А.				
Сем. пол.	X	Романова М.С.				
Родился	12.10.56	Миронов Р.И.				
Пол	М	Яковлев А.И.				
Фамилия	Потапов Д.П.	Кулакова М.И.				
Детей	3					

Рис.4.3

Он состоит из двух частей. В левой половине представлены вертикально (в режиме CHANGE) все перечисленные поля базы данных, но в иной последовательности — сначала идут более значащие для нас в данном случае поля, содержащие сведения о числе детей, средней зарплате и материальной помощи. В правой половине отображено одно только поле фамилий, но во всю свою длину. Такое представление информации здесь гораздо удобнее для пользователя. В правой части окна он как бы имеет только оглавление базы (фамилии работников), а в левой — полное ее раскрытие. Перемещение курсора из одной половины в другую можно осуществлять мышью или клавишами Ctrl-N.

Как получен такой экран? Сначала в исходном BROWSE-окне удаляются ненужные поля (все поля, кроме поля фамилий). Для этого маркер мыши устанавливается на соответствующий вертикальный разделитель, нажимается кнопка мыши, и разделитель буксируется влево до полного скрытия ненужного поля.

Вертикальные линии у правой границы окна — это видимые остатки скрытых полей. Наоборот, поле фамилий точно таким же образом расширяется до желаемой ширины. В самой команде BROWSE ему для экономии места на экране отведено 12 символов, но фактическая длина поля 25 символов. Теперь маркер мыши устанавливаем на специальный значок в левом нижнем углу окна (на рисунке не показан) и буксируем его вправо. При этом открывается левая половина окна, где поля расположены вертикально. Такую возможность предоставляет включение в команду BROWSE опции LEDIT. Далее изменяем порядок следования полей. Для этого также мышью буксируем имя соответствующего поля на нужное место.

Естественно, что переконфигурация окна может быть выполнена и в другом порядке. Все сделанные настройки при желании могут быть сохранены, если в команду BROWSE включены опции LAST или PREFERENCE.

Как видим, мышь предоставляет нам огромные удобства. Эти же действия в принципе возможно осуществить и с клавиатуры, но для этого необходимо обратиться к системному меню BROWSE. Хотя это меню может быть доступно и в прикладных программах, отечественный пользователь, скорее всего, не захочет работать с ним в его оригинальной форме на английском языке. Вместе с тем в BROWSE-меню есть одна возможность, которая мышью не реализуется — это переключение режима синхронного/асинхронного перемещения записей в обеих половинах разделенного окна. По умолчанию перемещение курсора в одной половине влечет синхронное же движение записей в другой. Режим асинхронного движения может быть установлен опцией NOLINK непосредственно в команде BROWSE.

Кроме управления форматом предъявления данных можно управлять и конфигурацией (размером и местоположением) самого окна, в которое помещена база, если при описании окна употреблены необходимые опции. Подробнее системные средства FoxPro, работа с окнами, а также более сложные примеры использования команды BROWSE будут разобраны далее.

Замечание к нотации примеров. Все тексты примеров изображаются так, как будто они являются фрагментами программ, и при этом длинные команды разбиваются на части приемлемого размера с помощью знака ";". Поскольку еще не рассматривались средства создания и исполнения командных файлов, эти команды следует вводить в командном окне, можно опуская ";". Кроме того, не следует повторно вводить команды, если нужный результат уже ранее достигнут. Так, нет необходимости снова открывать файл базы данных, если он уже открыт. Убедиться в этом можно, например, по виду статус-строки.

4.2. CHANGE/EDIT-окно

Команда CHANGE предъявляет на редактирование перечисленные поля записей базы данных в указанных границах и для заданных условий.

■ CHANGE [**<границы>**] [FOR **<условие>**]
[WHILE **<условие>**] [FIELDS **<поля>**] [опции]

Команда позволяет редактировать отдельные записи в базе данных.

<Границы>, WHILE- и FOR-условия ограничивают работу команды только заданным диапазоном записей, FIELDS **<поля>** — перечнем обрабатываемых полей.

Опции — набор режимов по составу и действию почти полностью аналогичны опциям команды BROWSE. Отсутствуют только опции NOLGRID/NORGRID. При разделении окна здесь также можно вывести данные в формате BROWSE.

Используя команду, можно редактировать любое поле записи, перемещаться к другим записям, добавлять новые записи. Отсутствие границы и условий означает, что доступны все записи. Отсутствие ключевого слова FIELDS указывает на обработку всех полей записи.

П р и м е р. Выдать на редактирование поля FAM (Фамилия) и SEM (Семейное положение) всех записей файла KADR.DBF, для которых значение поля DET>1 (детей больше одного).

```
.USE kadr
.CHANGE FIELDS fam,sem FOR det>1
```

П р и м е р. Выдать на редактирование все записи, относящиеся к мужчинам (FOR pol='M'), а в них поля FAM, DTR, POL, SEM. При вводе в поле POL установить входной контроль данных, допускающий ввод только символов "М" и "Ж". При ошибочном вводе предусмотреть вывод предупреждения "Только М или Ж". Аналогичный контроль ввести и в поле

SEM, где разрешены только символы "Б", "Х", "Р". Кроме того, сформировать вычисляемое поле PEN (Пенсия), в котором для людей пенсионного возраста появляется слово "Пенсионер".

```
.USE kadr
.SET DATE GERMAN
.CHANGE FOR pol='М' TITLE 'Сегодня - '+DTOC(DATE());
FIELDS fam :H='Фамилия',;
      dtr :H='Родился',;
      pol :H='Пол' :V=pol='М'.OR.pol='Ж';
      E='Допустимы значения - М или Ж',;
      sem :H='Сем. положение' :V=INLIST(sem, 'Б', 'Х', 'Р');
      E='Вводить только Б (в браке), Х (холост)'+;
      'или Р (разведен)',;
      pen=IIF(GOMONTH(dtr, 12*60)<=DATE(), 'Пенсионер', '');
      :H='Пенсия'
```

Ввод данных в поле POL контролируется опцией :V, где оно проверяется на равенство буквам "М" или "Ж" с помощью логической связки OR. Более удобным в таких случаях (в особенности, если возможно несколько альтернативных значений) использование не логических отношений, а функции INLIST(), которая возвращает значение "Истина", если поле (в данном случае SEM) совпадает с одним из перечисленных далее аргументов (см. гл.16).

Для определения факта достижения пенсионного возраста используется функция GOMONTH(), которая возвращает новую дату, являющуюся суммой аргумента DTR (Дата рождения) и 12*60 месяцев (мужчины достигают пенсионного возраста в 60 лет). Если полученная дата меньше или равна текущей (DATE()), значит это "Пенсионер", если нет, в столбце PEN выводится пустая строка ("").

Кроме того, в заголовке CHANGE-окна выводятся слово "Сегодня" и текущая дата. Чтобы можно было их сочетать в одной строке, функцией DTOC() дата преобразуется в символьный тип и соединяется (+) со словом "Сегодня".

При наличии условий в команде CHANGE в окно редактирования выводится не столько записей, сколько удастся разместить, а только одна текущая (рис. 4.4).

Сегодня - 10.07.92	
Фамилия	ЯКОВЛЕВ А.И.
Родился	10.12.30
Пол	М
Сем. положение	Б
Пенсия	Пенсионер

Рис.4.4

Команда CHANGE полностью идентична команде EDIT.

Замечание к вводу дат. При вводе данных в поля/переменные (командами BROWSE/CHANGE, READ) типа дата мы обнаружим, что в такие поля разрешается вводить лишь содержательные данные в допустимом диапазоне. Не разрешается вводить пустую дату ({ . . . }), хотя дату такого вида мы можем видеть при первоначальном заполнении базы. В поля типа дата вводить пробелы вообще не удастся, а ввод, например, строки вида 00.00.00 не допускают средства внутреннего контроля FoxPro. Между тем это очень важно, так как бывает, что дата не известна или же она еще не установлена. В этом случае неплохо оставить ее пустой, а не заполнять временно бессмысленной, хотя формально и разрешенной датой. Возможность очистить такое поле существует — это использование клавиш удаления поля Ctrl-Y.

Окна редактирования могут быть еще более адаптированы под любые запросы заказчика с помощью форматного файла или путем непосредственного использования команд @ ... SAY ... GET. К этим вопросам мы вернемся ниже.

Глава 5. ПЕРЕМЕЩЕНИЯ В БАЗЕ ДАННЫХ

При работе с базой данных необходимы средства перемещения внутри нее. Запись, на которой находится указатель записей, является текущей, и только к ней в данный момент возможен непосредственный доступ.

Имеется несколько разновидностей команд, изменяющих положение указателя записей:

- | | |
|---------------|---|
| ■ GO TOP | — переход к самой первой записи файла; |
| ■ GO BOTTOM | — переход к самой последней записи; |
| ■ GO <вырN> | — переход к записи с указанным в <вырN> номером; |
| ■ SKIP <вырN> | — переход к записи, отстоящей от текущей на указанное в <вырN> число записей. |

В последней команде <вырN> может быть и отрицательным, что означает движение указателя назад. SKIP без параметра идентичен SKIP 1 (переход на следующую запись). Все вышеперечисленные команды могут иметь дополнительный параметр

IN <область>

указывающий для базы данных, из какой области должна выполняться команда. Если он опущен, имеется в виду текущая рабочая область.

Для контроля положения указателя и наличия записей в файле предусмотрены функции:

- | | |
|-------------------------|---|
| ■ RECNO([<область>]) | — указывает номер текущей записи; |
| ■ RECCOUNT([<область>]) | — выдает общее число записей в файле базы данных, включая записи, помеченные к удалению; |
| ■ EOF([<область>]) | — функция конца файла. Она истинна (.T.), если конец достигнут, и ложна (.F.) в противном случае; |
| ■ BOF([<область>]) | — то же, но для начала файла. |

Необязательный параметр <область> указывает, для какой рабочей области запрашивается значение функции. По умолчанию текущая область.

Функции в FoxPro имеют характерный синтаксис — скобки, даже если никакого аргумента нет и они остаются пустыми.

Глава 6. ПРОСМОТР ДАННЫХ

Просмотр данных в FoxPro осуществляется очень близкими по смыслу и синтаксису командами LIST и DISPLAY:

■ **DISPLAY** [<границы>] [<поля>]
 WHILE [<условие>] [**FOR** <условие>]
 [**OFF**] [**TO PRINT**/**TO FILE** <файл>]

Здесь:

OFF — указание на то, что номера записей не выводятся;
TO PRINT — результат команды выдается на принтер;
TO FILE <файл> — результат выдается в <файл>. Если не указать расширение имени, то оно будет TXT.

В качестве заголовка вывода командой выдаются имена полей базы данных. При заполнении экрана выполнение команды приостанавливается с индикацией указания "Нажмите любую клавишу". При нажатии просмотр может быть продолжен. После выполнения команды указатель записи перемещается на последнюю показанную запись. Записи, помеченные к удалению, команда выдает со звездочкой (если **SET DELETED OFF**).

Выдачу имен полей можно подавить командой

■ **SET HEADING OFF**

(восстановление — **ON**). Чтобы выдать мемо-поля, их имена нужно явно указать в списке **FIELDS**, иначе они выводятся просто столбцом "мемо". Команда

■ **SET MEMOWIDTH <вырN>**

определяет фактическую ширину строки для выводимого мемо-поля.

Команда **DISPLAY** без параметров осуществляет выдачу всех полей базы данных только одной текущей записи.

Примеры:

.USE kadr	
.DISPLAY ALL TO FILE kadr	- выдача всех записей в файл KADR.TXT
.DISPLAY fam WHILE tab#='890'	- выдача полей FAM всех записей до тех пор, пока не встретится запись с TAB='890'
.GO 4	- переход к четвертой записи
.DISPLAY REST FOR pol='Ж' .AND. sem='Б'	- выдача всех записей файла, начиная с четвертой, для всех женщин, состоящих в браке
.GO TOP	- переход в начало файла
.SET MEMOWIDTH TO 40	- установление длины вывода мемо-поля в 40 колонок
.DISPLAY fam,per OFF	- вывод из текущей (первой) записи поля FAM и мемо-поля PER. Номер записи не выводится

Команда с похожими функциями **LIST** не делает периодических остановок при выдаче данных, и по умолчанию область ее действия не текущая запись, а весь файл. Ввиду этого команда более пригодна для выдачи данных на принтер/файл. Выполнение команды **LIST** может быть инициировано в командном режиме нажатием клавиши F3, а команды **DISPLAY** — F8.

Приведенные команды удобны скорее для просмотра данных при отладке программ. Для включения их в программы они слишком грубы, и здесь лучше использовать более гибкие команды ? и @ ... SAY.

Глава 7. УДАЛЕНИЕ ДАННЫХ

В FoxPro имеется несколько команд удаления данных:

- **ERASE <файл>** — удаление любого не открытого в данный момент файла. Расширение имени обязательно. Совершенно аналогичные функции выполняет команда **DELETE FILE <файл>**.
- **ZAP** — удаление всех записей в активном файле базы данных с сохранением его структуры.
- **DELETE [<границы>] [WHILE <условие>] [FOR <условие>]** — пометка к удалению записей в указанных границах и/или отвечающих указанным условиям. **DELETE** без параметров помечает только одну текущую запись.
- **PACK [MEMO] [DBF]** — физическое удаление помеченных ранее записей и сжатие файла. После выполнения команды указатель записей устанавливается в начало базы. Если имеются открытые индексы, они перестраиваются. По умолчанию упаковывается как файл данных (**DBF**), так и файл мемо-полей (**FPT**). Если указан параметр **MEMO**, то упаковывается только **FPT**-файл, если **DBF** — то только **DBF**-файл.
- **RECALL [<границы>] [WHILE <условие>] [FOR <условие>]** — снятие пометок к удалению. **RECALL** без параметров действует только на одну текущую запись.

Удаление записей в базе данных выполняется в два этапа: сначала пометка записей на удаление (она возможна и в окнах редактирования нажатием клавиш Ctrl-T) командой **DELETE**, а затем их физическое уничтожение командой **PACK**. Если упаковка файла еще не произведена, можно спасти нужные записи командой **RECALL**.

Команду **PACK** имеет смысл применять не только для удаления записей, но и для сжатия мемо-полей (**PACK MEMO**). Дело в том, что даже при видимом уменьшении размера мемо-поля (например, в результате редактирования) уменьшается только доступная пользователю часть, но размер самого **FPT**-файла никогда не сокращается. Таким образом, при интенсивном обмене данными с участием мемо-полей вполне возможно недопустимое "разбухание" файла примечаний.

П р и м е р ы:

```
.USE kadr
.?RECNO(),RECCOUNT() - выдача номера текущей записи и общего их числа
  1      7
.GO 5 - переход к пятой записи
.SKIP -3 - возврат ко второй записи
.DELETE NEXT 3 - пометка к удалению записей 2,3,4
.RECALL RECORD 4 - снятие пометки с записи 4
.PACK - сжатие файла с возвратом в начало базы
.?RECNO(),RECCOUNT()
  1      5 - осталось 5 записей
```

Для иллюстрации работы FoxPro применена команда **?**, выдающая на экран перечисленные в ней выражения. Более подробно команда будет рассмотрена позже.

Команда сжатия **PACK** в базе реальных размеров выполняется медленно, и поэтому лучше ее делать один раз в день или даже раз в несколько дней. Чтобы помеченные к удалению записи до их уничтожения не участвовали далее в обработке данных, можно использовать команду

■ SET DELETED ON

При этом такие записи делаются как бы невидимыми для FoxPro и пользователя, за исключением случаев прямого на них указания. Например, команда **GO 20** установит указатель записей на двадцатую запись независимо от того, была или нет она помечена. По умолчанию принято значение **OFF**.

Глава 8. ИЗМЕНЕНИЕ ДАННЫХ

В FoxPro имеется возможность не только вручную редактировать данные, но и изменять их путем присвоений или вычислений.

■ **REPLACE** [<границы>] [**WHILE** <условие>] [**FOR** <условие>]
<поле1> **WITH** <выражение> [, <поле2> **WITH** <выражение> ...]
[**ADDITIVE**] [**NOOPTIMIZE**]

Эта команда осуществляет множественное изменение полей базы данных в соответствии с заданными выражениями, в установленных границах и при заданных условиях. Если отсутствует параметр <границы> или <условия>, изменена будет только текущая запись. Параметр **ADDITIVE** действует для мемо-полей и означает, что заданное <выражение> будет дописываться в конец поля. Если этот параметр опущен, то старое значение мемо-поля будет замещено <выражением>. Для иллюстрации применения этой команды создадим еще один файл базы данных по учету месячной выработки (зарплаты без вычетов) всех членов бригады номер 1. Назовем его BRIG1.DBF. Файл будет иметь два поля: TAB — табельный номер и VIR — выработка-зарплата.

Командой **CREATE brig** установим структуру файла (рис. 8.1).

Name	Type	Width	Dec
TAB	Numeric	3	0
VIR	Numeric	7	2

Рис.8.1

Пример. Предположим, что бригаде 1 (файл BRIG1.DBF) выделена премия в размере 20% зарплаты, а бригадиру (табельный номер 98) — еще 500 руб. Премия не начисляется тем рабочим, которые в данном месяце отработали менее одного дня (выработка менее 100 руб.). С учетом сказанного в файле необходимо увеличить значения всех полей VIR в 1.2 раза, если VIR>100, а в записи с TAB=98 прибавить еще 500.

```
.USE brig1  
.REPLACE vir WITH vir*1.2 FOR vir>100  
.REPLACE vir WITH vir+500 FOR tab=98
```

Начальное содержимое базы данных представлено на рис.8.2, а новое — на рис.8.3.

	TAB	VIR
1	98	2500.00
2	6	1400.00
3	13	54.00
4	468	2050.00

Рис.8.2

	TAB	VIR
1	98	3500.00
2	6	1680.72
3	13	54.00
4	468	2460.00

Рис.8.3

Практически (за некоторыми исключениями) только командой **REPLACE** в FoxPro можно изменять значения полей файла базы данных. В этом смысле она эквивалентна знаку равенства в операции присвоения для переменных. Буквально фраза <поле> **WITH** <выражение> соответствует присвоению <поле>=<выражение>.

В команде REPLACE можно делать сразу несколько присвоений, в том числе и одному и тому же полю. Присвоения тогда выполняются слева направо. Так, предыдущий пример может быть реализован одной, а не двумя командами REPLACE:

```
REPLACE vir WITH vir*1.2, vir WITH IIF(tab=98,vir+500,vir) FOR vir>100
```

Здесь сначала начисляется премия в 20%. Затем, если TAB=98, поле VIR замещается на VIR+500 и остается без изменений в противном случае. Анализ и замещение данных производятся с помощью функции IIF() (см. гл.16). При этом также ускоряется и процесс обработки данных.

Решение задачи можно упростить еще больше:

```
REPLACE vir WITH IIF(tab=98,vir*1.2+500,vir*1.2) FOR vir>100
```


Глава 9. ЛОКАЛИЗАЦИЯ И ПОИСК ДАННЫХ В БАЗЕ

Исключительно важную группу команд в FoxPro образуют средства выделения и поиска данных в базе данных.

9.1. Фильтрация данных

Хотя в команде BROWSE, например, имеется возможность осуществить отбор записей из базы с помощью опций FOR и KEY, в других командах это может оказаться невозможным или неудобным. В FoxPro предусмотрена специальная команда вида

■ SET FILTER TO <условие>

которая позволяет установить FOR-условие для всех без исключения команд обработки данных. Здесь <условие> указывает, какие именно записи могут быть доступны для обработки. Например, команда

```
SET FILTER TO fam='IB'
```

сделает доступными для обработки только записи, в которых фамилия сотрудника начинается с букв "ИБ".

Команда SET FILTER действует исключительно на ту базу, которая открыта и активна в данный момент. То есть для каждой базы данных может быть установлен свой фильтр записей.

Команда SET FILTER TO без параметра снимает все ограничения на предъявление записей из текущей базы.

Установление фильтра имеет одну особенность — он начинает действовать только в случае, если после команды SET FILTER TO <условие> произведено хоть какое-то перемещение указателя записей в файле базы данных.

При возможности следует применять гораздо более быстрый способ фильтрации с использованием индексного файла. Для этого используется параметр KEY в команде BROWSE, или индексация с FOR-условием, или технология Rushmore (см. команду INDEX).

В FoxPro имеются разнообразные команды поиска записей, которые реализуют как последовательный, так и ускоренный алгоритм.

При последовательном поиске выполняется сплошной перебор записей файла базы данных до установки указателя записей на искомую запись.

9.2. Последовательный поиск

Начальный поиск

Следующая команда осуществляет последовательный поиск одной самой первой записи в базе данных, удовлетворяющей заданному FOR-условию, среди записей, находящихся в заданных границах, и до тех пор, пока соблюдается WHILE-условие (если есть).

■ LOCATE FOR <условие> [<границы>] [WHILE <условие>]

В случае, если границы и WHILE-условие отсутствуют, поиск ведется во всем файле, начиная с первой записи.

При успешном поиске указатель записей устанавливается на найденную запись, функция RECNO() равна номеру этой записи, а функция FOUND(), оценивающая результат поиска, возвращает значение "Истина" (.T.). При неудачном поиске функция RECNO() равна числу записей в базе плюс 1, FOUND()=.F., а функция достижения конца файла EOF() возвращает .T.

Продолжение поиска

Команда, которая продолжает поиск записей, начатый ранее командой LOCATE, приведена ниже:

■ CONTINUE

Если командой LOCATE или CONTINUE не было найдено нужных записей, указатель записей устанавливается на нижнюю границу поиска (если она введена в команде) или на конец файла (EOF()=Т.).

Результатом применения команд (если SET TALK ON) являются сообщения о номере каждой найденной записи или/и достижении границы поиска.

П р и м е р. В файле KADR (его содержимое представлено при описании команды BROWSE) необходимо найти все записи о женщинах, т.е. те записи, в которых POL='Ж'. Вводимые команды и реакции системы изображены ниже (найденны записи с номерами 3 и 5).

```
.USE kadr
.Locate FOR pol='Ж'
Record -> 3           (Запись 3)
.CONTINUE
Record -> 5           (Запись 5)
.CONTINUE
End of Locate scope   (Конец границы поиска)
```

Кроме рассмотренных команд имеется полезная функция поиска

■ LOOKUP(<поле1>,<выр>,<поле2>)

Функция ищет первое вхождение искомого <выражения> в указанном <поле 2> активной базы данных и возвращает значение <поля 1> из той же базы. Если файл индексирован и индекс открыт, поиск ведется ускоренным методом, если нет — последовательным (подобно команде LOCATE). Если поиск оказался безуспешным, функция возвращает пустую строку, а указатель записей становится ниже последней записи базы (EOF()=Т.).

П р и м е р. В базе KADR.DBF с помощью функции LOOKUP() ищется в поле FAM первая фамилия, начинающаяся на букву 'П', и выводится ее табельный номер. Кроме того, для проверки в команде ? выводится и сама фамилия, и номер записи (FAM и RECNO()). Результаты работы команды приведены справа после знаков &&. Видим, что сначала выводится табельный номер 98 (ПОТАПОВ Д.П. — запись номер 2). После открытия индексного файла KADRFAM.IDX (индексирование сделано по полю FAM) выводятся уже данные для ПОПОВА А.А. (запись номер 4), поскольку по алфавиту он стоит выше ПОТАПОВА Д.П.

```
.USE kadr
.? LOOKUP(tab,'П',fam),fam,RECNO()    && 98 ПОТАПОВ Д.П. 2
.SET INDEX TO kadrfam
.? LOOKUP(tab,'П',fam),fam,RECNO()    && 234 ПОПОВ А.А. 4
```

А сейчас рассмотрим понятие индексирования и соответствующие команды.

9.3. Индексирование баз данных

Важнейшим элементом любой системы управления базами данных является наличие средств ускоренного поиска данных, поскольку поиск — самая распространенная операция в системах обработки данных. Этот механизм обычно реализуется введением так называемых индексных файлов (индексов). Они имеют расширение имени IDX/CDX.

Если файл проиндексирован, команды DISPLAY, EDIT, BROWSE, SKIP, REPLACE и все другие команды, связанные с движением в файле базы данных, перемещают указатель записей в соответствии с индексом, а не с

физическим порядком расположения записей. Так, команды GO TOP и GO BOTTOM устанавливают указатель записей не на первую (номер 1) и последнюю физические записи, а на начальную и конечную записи индекса соответственно. Ввиду этого для экономии времени индексы лучше отключать, когда они не нужны.

Один файл базы данных может быть проиндексирован по нескольким полям и иметь любое число индексов (индексных файлов), которое ограничено только дисковой памятью компьютера. Такие файлы содержат информацию о расположении записей файла базы данных в алфавитном, хронологическом или числовом порядке для того поля/полей, по которому выполнено индексирование. Допускается индексирование и по логическим полям.

Аппарат индексирования является важнейшим инструментом любой реляционной СУБД. И хотя практически все действия над данными могут быть осуществлены и без участия индексов, совершенно немыслимо их игнорирование при создании реальных информационных систем. Только использование индексов позволяет достичь приемлемых скоростных характеристик обработки данных, поскольку поисковые операции (присутствующие прямо или косвенно) используются в программах очень широко. Однако за все надо платить. Сами индексные файлы занимают некоторое место на диске. Размер индексного файла сравним с объемом дискового пространства, занимаемого полем базы данных, по которому было произведено индексирование. Таким образом, например, если база проиндексирована по всем полям, суммарный размер всех индексов будет близок (или больше) к размеру всей базы данных. Кроме того, замедляются операции ввода/редактирования данных в базе, поскольку при дополнении ее новой записью индексный файл должен быть автоматически перестроен в соответствии с новыми или измененными данными.

В FoxPro можно создать два типа индексных файлов:

- Обычный индексный файл. Он имеет расширение имени IDX и содержит один индексный ключ. Его можно также назвать одноиндексным файлом.
- Мультииндексный файл с расширением имени CDX. Такой файл может хранить сразу несколько индексных выражений и является по существу соединением нескольких простых индексных файлов. Каждый отдельный индекс в нем будем называть ввиду отсутствия подходящего термина словом "тег", взятым непосредственно из технической документации (TAG — этикетка). Каждый тег имеет свое имя.

Мультииндексные файлы могут быть двух видов: структурный мультииндексный файл с именем, совпадающим с именем базы данных, и обычный мультииндексный файл с произвольным именем.

Структурный файл всегда автоматически открывается вместе со своей базой. Его нельзя закрыть, хотя можно сделать неглавным.

Использование мультииндексных файлов предпочтительнее в случае одновременной работы со многими файлами, поскольку отодвигает нас от границы DOS на 99 открытых файлов, а также несколько ускоряет доступ к файлам (ведь их меньше в директории) и, кроме того, позволяет программисту легче контролировать свою файловую среду.

Индексирование выполняется следующей командой:

```
■ INDEX ON <выр> TO <IDX-файл>/TAG <имя тега> [OF <CDX-файл>]  
[FOR <условие>] [COMPACT] [DESCENDING] [UNIQUE] [ADDITIVE]
```

В FoxPro с версии 2.0 принята новая более компактная и более быстрая в обработке структура индексного файла. Мультииндексный файл всегда компактный. Обычный (IDX) индексный файл строится обычным образом,

если не указан параметр COMPACT. Сделано это для совместимости со старыми индексными файлами из СУБД FoxBASE и FoxPro-1.x. Все новые индексные файлы, конечно, следует создавать только компактными.

Опции команды:

<выр> — индексный ключ-выражение. Его длина может достигать 100 символов для IDX-файлов и 254 для CDX-файлов. Обычно ключ означает имя поля, по которому нужно упорядочить файл. Ключ может быть и составным — из нескольких полей. Он может быть и функцией, в том числе и ПФ, полей и переменных.

TO <IDX-файл> — задает имя одноиндексному файлу.

TAG <имя тега> [OF <CDX-файл>] — задает имя тега в мультииндексном файле. Если в команде присутствует фраза OF <CDX-файл>, то мультииндексный файл получит указанное имя. Если нет, то будет создан структурный мультииндексный файл с именем, совпадающим с именем базы данных. Команда может использоваться как для создания нового мультииндексного файла, так и для дополнения уже существующего CDX-файла новым тегом.

FOR <условие> — эта опция является важным расширением команды индексирования. Она устанавливает режим отбора в индекс только тех записей базы данных, которые отвечают заданному <условию>. При наличии такого, действующего, как фильтр, индекса доступ к нужным данным осуществляется исключительно быстро.

COMPACT — с этой опцией будет создан компактный IDX-файл. Целесообразно использовать только компактные индексы. Возможность создавать обычные индексы оставлена лишь для совместимости с предыдущими версиями пакета.

DESCENDING — индексирование будет выполнено по убыванию. Этот режим можно указывать только для мультииндексных CDX-файлов. Для IDX-файлов индексирование всегда осуществляется по возрастанию, однако параметр DESCENDING можно включить в команды открытия индексов любого типа, независимо от того, какой закон был указан в команде индексирования. По умолчанию индексирование выполняется по возрастанию.

UNIQUE — означает, что, если в базе данных встречаются записи с одинаковым значением ключа, все такие записи, кроме первой, игнорируются (не включаются в индекс). Этим процессом можно управлять также с помощью команды SET UNIQUE.

ADDITIVE — вновь создаваемые индексные файлы не закроют уже открытые к этому моменту. По умолчанию вновь создаваемые индексы закрывают все ранее открытые индексы для текущей базы данных.

Рассмотрим несколько примеров индексирования.

П р и м е р. Пусть мы хотим упорядочить базу KADR в порядке возрастания табельных номеров. Тогда необходимо создать индексный файл по полю TAB. Назовем его KADRTAB.IDX (расширение IDX можно не указывать).

```
.USE kadr
.INDEX ON tab TO kadrtab.idx COMPACT
.LIST tab,fam
Record #   TAB   FAM
3         6     КУЛАКОВА М.И.
1        13     СИДОРОВ П.С.
7        54     ЯКОВЛЕВ А.И.
2        98     ПОТАПОВ Д.П.
4       234     ПОПОВ А.А.
6       468     МИРОНОВ Р.И.
5       890     РОМАНОВА М.С.
```


Как видно, команда LIST показала записи именно в желаемом, а не в фактическом порядке. Файл базы KADR.DBF никакому изменению не подвергся, но всеми перемещениями указателя записей управляет теперь индексный файл KADRTAB.IDX.

Каково содержимое индексного файла? Хотя в FoxPro не предусмотрена какая-либо возможность непосредственного доступа к индексу, мы можем просмотреть его во внутреннем текстовом редакторе FoxPro (с помощью команды MODIFY FILE KADTRAB.IDX). Тогда мы увидим, что кроме технической информации в заголовке файл KADRTAB.IDX будет содержать значения ключевого поля TAB индексируемой базы данных, расположенные в порядке возрастания, и (в закодированной форме) фактические номера этих записей в базе данных. В нашем случае

3 — 6, 1 — 13, 7 — 54, 2 — 98, 4 — 234 и т.д.

П р и м е р. Рассмотрим включение FOR-условия в команду индексирования. Пусть требуется создать индексный файл KADRPOL.IDX, содержащий ссылки только на те записи базы KADR.DBF, которые соответствуют всем мужчинам (pol='M'), причем упорядоченные в алфавитном порядке фамилий.

```
.USE kadr
.INDEX ON fam TO kadrpol FOR pol='M' COMPACT
.LIST FAM, POL
Record #   FAM                                POL
      6   МИРОНОВ Р.И.                      М
      4   ПОПОВ А.А.                        М
      2   ПОТАПОВ Д.П.                      М
      1   СИДОРОВ П.С.                      М
      7   ЯКОВЛЕВ А.И.                      М
```

Видим, что содержание и порядок предъявления записей из базы KADR.DBF отвечают желаемым.

Если порядок предъявления фамилий безразличен и по ним не предполагается поиск, ключевое поле (в данном случае FAM) можно опустить, заменив его пробелом в команде индексирования

```
.INDEX ON ' ' TO kadrpol FOR pol='M' COMPACT
```

При этом индексный файл будет гораздо меньших размеров.

Если индексный файл был уже создан, его нужно открыть при внесении новых записей или редактировании старых (если затрагиваются индексные поля) и, конечно, если предполагается индексный поиск. Индексные файлы могут быть открыты совместно с открытием своей базы данных командой:

```
■ USE [<DBF-файл>] / [IN <область>]
  [AGAIN] [NOUPDATE] [INDEX <список индексных файлов>]
  [ORDER [<вырN> / <IDX-файл>] / [TAG] <имя тега>]
  [OF <CDX-файл>] [ASCENDING / DESCENDING]]]
  [ALIAS <псевдоним>]
```

Если указан параметр IN <область>, база откроется не в текущей, а в указанной рабочей <области>, но автоматический переход в <область> при этом не произойдет. По умолчанию база открывается в текущей области. Если не осуществлен переход в какую-либо область, имеется в виду область 1 (или А). Подробнее понятие рабочей области будет рассмотрено позже.

Опция AGAIN позволяет открыть открытый уже файл базы данных в другой рабочей области. В области, где он был открыт ранее, такой файл может быть доступен в режиме только-чтения. Однако каждый индекс может быть открыт одновременно только для одной из областей.

Опция INDEX <список индексных файлов> указывает перечень

открываемых индексных файлов. Здесь важен порядок перечисления индексов. Первый в списке индекс считается главным, если не использована опция ORDER. Понятие главного индекса будет введено ниже.

Параметр ALIAS [<псевдоним>] устанавливает для базы кроме собственного еще любое другое имя. Если разным базам дать одинаковый псевдоним, далее они могут под этим именем обрабатываться в одной и той же программе.

Параметр NOAPDATE указывает на то, что база будет открыта только для просмотра (редактирование запрещено).

Открытие файла всегда устанавливает указатель записей на его первую запись.

Новая команда USE открывает новый файл базы данных одновременно с закрытием старого в текущей области. Команда USE без имени файла закрывает соответствующий файл базы данных и все сопровождающие его вспомогательные файлы (типа IDX и др.) в текущей или другой (если она указана) области.

Остальные опции управляют открытием индексных файлов и указанием главного индекса. Они полностью идентичны соответствующим параметрам команды SET INDEX, которая открывает только индексы (база должна быть уже открыта):

- SET INDEX TO [<список индексных файлов>
[ORDER <вырN>/<IDX-файл>/[TAG] <имя тега>
[OF <CDX-файл>] [ASCENDING/DESCENDING]] [ADDITIVE]

Команда открывает перечисленные через запятую одно- и мультииндексные файлы (кроме структурного мультииндексного файла, который открывается автоматически вместе с базой данных). По умолчанию открытый первым IDX-файл становится главным индексом. Открытый первым CDX-файл не изменяет порядка обработки данных (не устанавливает главного индекса).

Опции команды:

ORDER <вырN> — опция указывает номер главного индексного файла среди перечисленных в <списке> открываемых IDX-файлов или среди тегов CDX-файла (номер по порядку создания), если нас не устраивает назначение главного индекса по умолчанию. Указание опции ORDER без аргумента или задание параметра <вырN>=0 означает, что, хотя индексы откроются, главный индекс не будет назначен.

ORDER <IDX-файл> — главный индекс задается указанием имени индексного файла.

ORDER [TAG] <имя тега> OF <CDX-файл> — главный индекс задается именем тега.

ASCENDING/DESCENDING — определяет порядок использования индекса (по возрастанию/убыванию), даже если при его создании был использован противоположный закон.

ADDITIVE — открытие новых индексов не закрывает старые.

Команда SET INDEX TO без параметра закрывает все индексные файлы, кроме структурного, для текущей базы. Такое же действие осуществляет команда

- CLOSE INDEX

Ускоренный поиск

Индексный файл не только упорядочивает базу данных для просмотра, но и ускоряет поиск в ней по ключу, заданному в индексе, если пользоваться командой

■ SEEK <выражение>

Команда применяет специальный алгоритм ускоренного поиска, в котором база просматривается не сплошь, а в соответствии с информацией, содержащейся в индексе.

При наличии индекса сначала именно в нем, а не в самой базе ведется поиск номера записи с указанным в команде SEEK значением выражения в индексном поле. При этом поиск в индексе выполняется не последовательно, а скачками (так называемый двоичный поиск), что позволяет быстро локализовать номер нужной записи. Только после этого указатель записей устанавливается на искомую запись в основной базе данных.

Команда SEEK разыскивает только одну первую запись, в которой в индексном поле наблюдается <выражение>, т.е. когда <поле>=<выражение>, и устанавливает на нее указатель записей.

П р и м е р. Проведем в базе KADR.DBF поиск записи с табельным номером 234.

```
.USE kadr INDEX kadrtab
.SEEK 234
.DISPLAY tab,fam
Record #   TAB      FAM
         4   234     ПОПОВ А.А.
```

Функции RECNO(), FOUND(), EOF() реагируют на результаты поиска командой SEEK точно так же, как и командами LOCATE и CONTINUE. Если поиск удачный, RECNO() равно номеру найденной записи, FOUND()=.T., EOF()=.F.; если нет, RECNO() равно числу записей в базе данных плюс единица, FOUND()=.F., EOF()=.T.. Все это относится и к индексам с FOR-условием.

Для индексированных баз существует модификация функции указания номера записи с аргументом нуль — RECNO(0), которая в случае неудачного поиска возвращает номер записи, имеющей самое близкое следующее значение к ключу поиска, заданному в команде SEEK. Используя этот номер, можно затем перейти в указанную запись. Однако если действует команда

■ SET NEAR ON

то в случае неудачного поиска указатель записей сразу установится не на конец файла, а на эту близкую запись. По умолчанию SET NEAR OFF.

Это предоставляет инструмент ускоренного поиска по ключу, заданному приблизительно или даже частично неправильно. Например, задана фамилия с неверными инициалами или окончанием. Часто такая ситуация встречается при поиске в числовых полях. Пусть в базе KADR.DBF нужно найти запись, где средняя зарплата равна 6000 руб. Ввиду того что, возможно, никто не получает именно такую зарплату, поиск окажется неудачным, хотя и есть зарплаты, близкие к этой цифре. Если же мы имеем возможность позиционировать указатель на записи с ближайшим значением, то, вызвав затем какое-нибудь средство просмотра данных (например, команду BROWSE) и пролистав данные в базе вблизи найденного места, мы получим возможность все-таки найти и отобрать подходящие записи.

```
.USE kadr
.INDEX ON szar TO kadrzar COMPACT
.SET NEAR ON
.SEEK 6000
.BROWSE
```

При этом мы не будем стеснены в возможности перемещаться в базе данных. Такой механизм называется "мягким" или приблизительным поиском в отличие от обычного, точного поиска.

База KADR.DBF проиндексирована по полю SZAR (средней зарплате) — файл KADRZAR.IDX. Похожий результат даст использование команды BROWSE с FOR-условием. Например, команда

```
BROWSE FOR szar>=5900.AND.szar<=6100
```

предъявит на редактирование все записи, где SZAR находится в диапазоне от 5900 до 6100, однако одновременно мы утрачиваем доступ к записям за установленными пределами.

В FoxPro имеется очень полезная функция индексного поиска

■ SEEK(<выражение>[,<область>])

Она так же, как и команда SEEK, выполняет поиск записи в индексном файле и устанавливает на него указатель записей с возвращением значения .T., если поиск удачный, и .F. — если нет. Функция SEEK() заменяет комбинацию команды SEEK и функции FOUND(). Такое совмещение весьма полезно, поскольку обычно для того, чтобы предпринять какие-то дальнейшие шаги после поиска, все равно нужно убедиться в его успешности, функция FOUND() обычно включается в состав команды анализа выполнения условий IF. Кроме того, она допускает поиск в неактивной рабочей <области>, заданной числовым выражением.

П р и м е р. Пусть нужно указать фамилии родителей, у которых трое детей. Если их не окажется, вывести ничего не нужно.

```
USE kadr
INDEX ON det TO kadrdet COMPACT
IF SEEK(3)
  LIST fam,det WHILE det=3
ENDIF
```

Рассмотрим технику индексного поиска. Двоичный поиск, называемый еще поиском делением пополам, начинается с середины индексного файла. Берется его центральный элемент и сравнивается с ключом, указанным в команде/функции SEEK. Если значение <ключа> больше <индексного выражения> этого элемента, то поиск ведется в нижней части индекса, если нет — в верхней. В выбранной половине снова находится середина и определяется новая подобласть поиска. И так до тех пор, пока не будет зафиксировано искомое совпадение либо не будет выяснено, что нужных данных нет. В первом случае указатель записей базы данных устанавливается на запись с найденным в индексе номером, во втором — вырабатывается значение .T. ("Истина") для функции EOF(), индицирующей достижение конца файла (если SET NEAR OFF).

В случае, например, поиска записи с табельным номером 234 в базе KADR.DBF, индексированной по полю TAB, будут выполнены поисковые действия в индексном файле KADRTAB.IDX в последовательности, показанной на рис. 9.1.

Файл KADRTAB.IDX

Номер записи БД	Индексное выражение
3	6
1	13
7	54
2	98 — 1-й шаг
4	234
6	468
5	890

2-й шаг — 3-й шаг

Рис. 9.1

Вычисляется середина индекса (1-й шаг поиска). Так как здесь находится табельный номер 098 < 234, далее разыскивается середина нижней половины

индекса (2-й шаг). Поскольку $468 > 234$, то теперь розыск следует вести между табельными номерами 098 и 468 (3-й шаг). Именно здесь и находится искомый табельный номер 234. Результатом поиска является установка указателя записей БД на запись с номером 4 в файле KADR.DBF.

Предельное число сравнений при двоичном поиске может быть определено значением логарифма по основанию 2 — $\text{Log}(N)$, где N — число записей в базе. Это значит, что если база имеет 1024 записи, то в худшем случае потребуется 10 обращений к индексу для розыска нужной записи. Конечно, поиск может завершиться и быстрее, если искомое индексное выражение находится в середине какого-то более крупного отрезка индексного файла. В случае, если искомой записи нет, выполняется именно $\text{Log}(N)$ обращений к индексу.

Компактные индексные файлы устроены по-другому. Здесь ключевые данные из записей базы данных хранятся в строках не фиксированной, а фактической длины, примыкая вплотную друг к другу. Кроме того, повторяющиеся ключевые выражения и повторяющиеся их части хранятся в индексе в одном экземпляре. Это обеспечивает, как правило, существенно меньший размер компактного индекса относительно обычного. Сокращение размера индексного файла позволяет разместить большую, чем обычно, его часть в основной памяти компьютера, уменьшив обращения к диску, что ускоряет процесс обработки индексированной базы данных, несмотря на то, что алгоритм поиска более сложен.

Применение компактного индекса дает преимущества в скорости (по сравнению с обычным индексом), в особенности когда обрабатывается сразу несколько баз и индексов, и тем заметнее, чем больше их размер.

Индексный ключ в команде создания индексного файла может быть составным, включая имена нескольких полей, соединенных знаком "+". Однако поля должны иметь один и тот же тип или должны быть приведены к одному и тому же (обычно символьному) типу функциями преобразования типов. При этом нужно очень внимательно следить за видом окончательного индексного выражения, полученного после преобразования и сцепления данных из разных полей. Ведь именно оно запоминается в индексном файле и по нему ведутся поиск и упорядочение записей. Это относится и к заданию ключа поиска в команде SEEK, который по типу и виду должен отвечать именно индексному выражению, а не исходному значению индексируемого поля/полей базы.

Хотя предельно разрешенная длина индексного выражения в FoxPro — 100 символов (для CDX-файлов 254 символа), следует по возможности ограничивать его размер. Это сокращает время доступа к данным и величину индексного файла. Например, при сортировке записей по алфавиту часто бывает достаточно упорядочение их на глубину всего в три — четыре символа. Здесь (а также если индексируемое поле длиннее разрешенных 100 символов) при указании индекса можно воспользоваться функцией выделения в поле нужного количества знаков слева — `LEFT()`. Однако следует учитывать, что сокращение длины индексного ключа относительно длины поля означает, что данные, которые имеют различия, оставшиеся за пределами ключа, будут считаться в индексе одинаковыми.

Сокращение индексируемой длины поля одновременно означает такое же ограничение длины ключа поиска, который мы можем указать в команде/функции SEEK. Вообще, если при индексировании применено какое-то преобразование данных, это преобразование должно быть применено или учтено при задании ключа поиска.

П р и м е р. Пусть требуется упорядочить базу KADR.DBF по полу, а внутри — по фамилии, т.е. по полям POL и FAM. Назовем этот индексный файл именем POLFAM.IDX.


```
.USE kadr
.INDEX ON pol+fam TO polfam COMPACT
.LIST pol,fam
Record # POL FAM
3 Ж КУЛАКОВА М.И.
5 Ж РОМАНОВА М.С.
6 М МИРОНОВ Р.И.
4 М ПОПОВ А.А.
2 М ПОТАПОВ Д.П.
1 М СИДОРОВ П.С.
7 М ЯКОВЛЕВ А.И.
```

Видим, что команда LIST выдает записи в желаемом порядке. Сначала идут все фамилии женщин (буква Ж идет раньше буквы М), а затем мужчины, а затем все фамилии мужчин; фамилии расположены в алфавитном порядке.

Команда индексного поиска SEEK является аналогом команды LOCATE для последовательного поиска. Однако команде продолжения поиска CONTINUE нет индексного аналога. Причина здесь очевидна. После того как командой SEEK найдена первая нужная запись, розыск остальных записей, удовлетворяющих ключу поиска, является тривиальным. Следующая такая запись (если есть) находится в индексе непосредственно ниже найденной, и переход на нее может быть выполнен просто командой SKIP. Единственно следует предусмотреть, чтобы в отобранное множество не попали записи, идущие сразу под нужными.

Хотя для каждого из приведенных примеров тут же создавался свой индексный файл, на практике, конечно, все индексные файлы обычно создаются один раз и одновременно с их базами данных самим программистом. В программе при необходимости они только открываются.

Если индексный файл был ранее создан, но вы забыли его своевременно открыть и внесли какие-то изменения в базу, то необходимо его открыть и обновить командой

■ REINDEX

Обычно такая забывчивость при больших размерах базы обходится дорого. Обновление, как и создание нового индексного файла при уже заполненной базе, требует времени. Иногда все же целесообразно прибегать к временному отключению индексных файлов. Так, каждое дополнение проиндексированной базы новыми записями влечет перестройку всего индекса, что иногда может показаться слишком медленным. В этом случае имеет смысл отключить индекс в момент заполнения и сделать переиндексацию после окончания ввода целой группы записей.

В системах обработки данных часто возникает необходимость в сложном индексировании. При этом возможно использование практически любых встроенных функций FoxPro, а также процедур и функций, определенных пользователем.

П р и м е р. Пусть требуется базу KADR.DBF упорядочить таким образом, чтобы выявить наиболее нуждающихся материально. Критерием будем считать размер зарплаты и число членов семьи. Для этого проиндексируем базу по полю средней зарплаты, деленному на число детей, плюс единица (сам работник). Частное округляем до копеек (ROUND(szar/(det+1),2)). Кроме того, необходимо учесть, состоит ли человек в браке. Будем считать, что если да, то при прочих равных условиях он нуждается меньше, чем не состоящий в браке, поскольку обычно супруг/супруга работает. При индексировании это обстоятельство учитывается следующим образом. Если человек не состоит в браке (SEM#"Б"), полученное частное остается неизменным, в противном случае к нему прибавляется величина в 0.001 руб. (+IF(sem#"Б",0,.001)). Подсуммируемое значение настолько мало, что может повлиять только на порядок расположения тех записей, где сумма,

приходящаяся на одного члена семьи, одинакова.

Следующие команды реализуют поставленную задачу:

```
. USE kadr
. INDEX ON ROUND(szar/(det+1),2)+IIF(sem#'Б',0,.001) TO pom
```

Если вам предстоит создание сложного индекса и вы не уверены, что правильно задали индексный ключ в команде индексирования, следует проверить вид самого индекса. Для этого можно прибегнуть к просмотру IDX-файла непосредственно в текстовом редакторе FoxPro. Однако осмысленный анализ "на глаз" возможен только для простых IDX-файлов, где индексные ключи находятся практически в естественной форме. Для компактных и тем более для CDX-файлов такой анализ ввиду их сложности практически невозможен. Удобнее всего вывести индексное выражение в команде BROWSE в качестве вычисляемого поля. Кроме того, полезно также вывести и номер каждой записи. Для предъявления индексного выражения можно прямо включить в команду BROWSE индексный ключ из команды INDEX после слова ON, но гораздо проще воспользоваться функцией

■ KEY(<вырN>)

где <вырN> — номер индекса в команде открытия.

Полученное символьное выражение "наполняется" конкретным значением с помощью функции подстановки EVALUATE() (см. гл.16):

```
BROWSE FIELDS szar :H='Зарплата',;
det :H='Детей',;
sem, :H='Сем. полож.',;
i=EVALUATE(KEY(1)) :P='####.###' :H='Индекс',;
n=RECNO() :H='N записи'
```

Экран, предъявляемый командой BROWSE, показан на рис. 9.2.

Зарплата	Детей	Сем.пол.	Индекс	N записи
2500.00	2	X	833.330	6
2500.00	2	Б	833.331	5
1800.00	1	X	900.000	3
3780.00	2	P	1260.000	2
2050.00	0	Б	2050.001	4
2500.00	0	X	2500.000	1

Рис. 9.2

Анализируя значения поля базы данных SZAR, DET, SEM и вычисляемые поля N и I (номер записи и вид индекса), можно выяснить правильность индексирования. Можно даже провести эксперимент. При изменении зарплаты, числа детей или семейного положения запись должна изменить свое местоположение среди других в соответствии с установленным законом. Обратите внимание также, что у лиц, состоящих в браке, в третьем разряде после точки в поле I, как и следовало ожидать, стоит единица.

Этот же результат можно получить, прибегнув к символьному представлению ключа индексирования

```
INDEX ON STR(ROUND(szar/(det+1),2),8,2) +IIF(sem#'Б',' ','*') TO pom
```

Здесь, если человек не состоит в браке, в конец ключа добавляется пробел, если состоит — любой другой символ с большим кодом, например "*".

Прежде чем перейти к следующему материалу, необходимо сказать несколько слов об индексировании по убыванию в составном индексе. Хотя все ключевое выражение может быть построено (или открыто) как по убыванию, так и по возрастанию, отдельные его компоненты должны быть преобразованы, если мы хотим получить для них иной, чем для всего ключа, закон. Для

этого числовые поля удобно умножить на минус единицу. Поля типа дата не могут быть на что-то умножены, но можно их вычесть из заведомо большей даты. Строковые поля, хотя технически и можно трансформировать нужным образом, практически лучше не трогать. Пусть нужно проиндексировать базу KADR.DBF по полям SZAR и DTR таким образом, чтобы зарплата шла по возрастанию, а дата рождения — по убыванию, т.е. среди лиц, имеющих одинаковое значение поля SZAR, сначала расположить младших по возрасту. Такое индексирование осуществляет команда, где DTR вычитается из заведомо большей даты 01.01.2000 г.

```
.INDEX ON STR(szar,7)+STR({01.01.2000}-dtr,4) TO <имя индекса>
```

Управление индексами

Как уже говорилось, один и тот же файл DBF может иметь любое число индексов, и все они могут быть одновременно открыты командами SET INDEX или USE...INDEX. В нашем случае команда

```
.USE kadr INDEX kadrtab, polfam
```

открывает два индекса KADRTAB.IDX и POLFAM.IDX.

При вводе, удалении, редактировании записей все открытые индексные файлы будут соответствующим образом изменяться. Однако главным управляющим индексом, т.е. таким, в соответствии с которым при необходимости будет перемещаться указатель записей, может быть, конечно, только один. Им является индексный файл, открытый самым первым в команде (здесь KADRTAB.IDX).

В случае, если необходимо сделать главным другой индекс, используется опция ORDER либо команда

SET ORDER TO

```
■ [ <вырN1> / <IDX-файл> / [TAG] <имя тега> [OF <CDX-файл>]
  [IN <область>] [ASCENDING/DESCENDING]] [ADDITIVE]
```

Команда объявляет главный индекс/тег среди открытых IDX- или CDX-индексных файлов в текущей или указанной рабочей <области>. Опции команды совпадают с описанными выше для команды SET INDEX TO. Например, следующая команда сделает главным индекс POLFAM.IDX

```
.SET ORDER TO 2
```

Команда SET ORDER TO 0 или просто SET ORDER TO без параметра отключает все индексы от управления перемещением указателя записей. Теперь уже не будет главного индекса. Однако сами индексы остаются открытыми и чувствительными к изменениям в базе данных.

При необходимости имя главного индексного файла может быть выяснено с помощью функции

```
■ ORDER([<область>])
```

Функция возвращает имя главного индексного файла или главного тега из мультииндексного файла для текущей рабочей области или области, заданной номером или псевдонимом.

Рассмотрим ряд вспомогательных команд и функций для работы с мультииндексными файлами.

```
■ COPY INDEXES <IDX-файлы> / ALL [TO <CDX-файл>]
```

Команда копирует все (ALL) или перечисленные через запятую открытые <IDX-файлы> в структурный или указанный мультииндексный <CDX-файл>. Каждый такой файл становится тегом в CDX-файле с тем же самым именем,

что и IDX-файл. Если CDX-файла нет, он создается.

■ COPY TAG <тег> [OF <CDX-файл>] TO <IDX-файл>

Команда создает <IDX-файл> из поименованного <тега>, находящегося в указанном или (по умолчанию) структурном мультииндексном <CDX-файле>. Файл должен быть открыт.

■ DELETE TAG <тег1> [OF <CDX-файл 1>] [,<тег2> [OF <CDX-файл 2>]] ...

■ DELETE TAG ALL [OF <CDX-файл>]

Команды удаляют указанные или все (ALL) теги из открытого в текущей рабочей области CDX-файла. По умолчанию сначала рассматривается структурный CDX-файл (если он есть).

■ NDX(<вырN>[,<область>])

Функция выдает прописными буквами полное имя индексного файла, открытого в текущей или указанной рабочей <области>. <ВырN> указывает номер файла в команде открытия. Если файла нет, возвращается пустая строка.

■ CDX/MDX(<вырN1>[,<область>])

Функция возвращает имя открытого CDX-файла. Первым считается структурный CDX-файл (если есть). Имя такого файла всегда совпадает с именем самой базы. Далее идут обычные мультииндексные файлы в порядке их открытия. Если структурного CDX-файла нет, рассматриваются только обычные CDX-файлы. <ВырN1> — номер по порядку индексного файла, имя которого возвращается с учетом вышеказанного. Если файла с таким номером нет, возвращается нулевая строка.

■ SYS(22 [,<область>])

Функция возвращает имя главного индекса или главного тега в IDX/CDX-файле. <Область> задается только числом.

■ TAG(<[<CDX-файл>,>] <вырN1> [,<область>])

В соответствии с порядком открытия (<вырN1>) возвращает имя тега из <CDX-файла> или имя IDX-файла для базы данных из текущей или указанной рабочей <области>. Если опущено имя CDX-файла, функция возвратит имя IDX-файла (если есть), или имя тега структурного мультииндексного файла (если есть), или имя тега из другого CDX-файла.

Примеры работы с мультииндексными файлами в базе KADR.DBF.

Создание индексного тега по фамилии (полю FAM) в структурном мультииндексном файле KADR.CDX. Тег получает имя FAM:

```
.INDEX ON fam TAG fam
```

Создание индексного тега DET по убыванию значения поля DET (число детей) в мультииндексном файле KADRUB.CDX:

```
.INDEX ON -det TAG det OF kadrub
```

Создание тега POLFAM в PFKADR.CDX по выражению POL+FAM. Одновременно индекс KADRUB.IDX закрывается:

```
.INDEX ON pol+fam TAG polfam OF pfkadr
```

Просмотр имен CDX-файлов командой DIR и с помощью функции CDX():

```
.DIR *.cdx                && KADR.CDX KADRUB.CDX PFKADR.CDX
.? CDX(1),CDX(2)         && KADR.CDX PFKADR.CDX
```

Просмотр имен всех тегов и выявление главного тега:


```
?.TAG(1),TAG(2),SYS(22)  && FAM  POLFAM  POLFAM
```

Копирование тега FAM из KADR.CDX в KADRFAM.IDX:

```
.COPY TAG fam TO kadrfam
```

Создание индексного файла ZAR.IDX:

```
.INDEX ON szar TO zar
```

Дополнение мультииндексного файла KADRUB.CDX тегами, образованными из всех открытых IDX-файлов. В данном примере, тегом ZAR из файла ZAR.IDX:

```
.COPY INDEXES ALL TO kadrub
```

Предупреждение. При работе с индексированными базами данных следует быть очень внимательным в случае изменения полей базы, по которым выполнено индексирование. После изменения значения такого поля командой REPLACE индекс немедленно перестроится и положение текущей записи среди других будет уже иное, чем ранее. Следствием этого может явиться получение неправильного результата, например некоторые записи базы данных, включенные в область действия команды, могут быть не обработаны (пропущены). Чтобы этого избежать, целесообразно отключение индексов от управления перемещением указателя записей командой SET ORDER TO 0 либо (если изменяется много данных) даже полное их отключение командой SET INDEX TO с последующим восстановлением командой REINDEX.

Индексирование не только логически упорядочивает записи базы данных, но позволяет выполнить и физическую сортировку данных. Для этого только надо скопировать проиндексированную базу в новый файл командой (COPY TO <новый файл>). Далее мы рассмотрим и специальную команду сортировки (SORT), которая сразу создает новый файл.

Кроме перечисленных в FoxPro реализована очень сильная команда отбора данных из языка SQL — SELECT (не путать с командой SELECT перехода в рабочую область). Эта команда будет рассмотрена отдельно.

9.4. Технология Rushmore

В FoxPro с версии 2.0 поиск и фильтрация данных для команд, содержащих FOR-условие, также могут быть реализованы с использованием индексов. При этом доступ к данным выполняется в несколько раз быстрее, чем ранее (в предыдущих версиях пакета), со скоростью, близкой к скорости индексного поиска. И выигрыш этот тем ощутимее, чем больше база данных.

Оптимизация стала возможной благодаря созданию разработчиками пакета эффективного способа доступа к данным, названного ими технологией Rushmore. Для его использования в прикладной программе необходимо наличие открытых индексных файлов с индексным выражением, присутствующим в FOR-условии. Это условие называется оптимизируемым выражением.

При этом эффективно используются индексные файлы любых типов — компактные и нет, IDX- и CDX-файлы. Исключением являются индексы, сами построенные с опциями FOR или UNIQUE, а также команды, содержащие опцию WHILE.

Ниже перечислены команды, в которых может использоваться новый метод. Эти команды допускают опцию FOR : AVERAGE, BROWSE, CALCULATE, CHANGE, COPY TO, COPY TO ARRAY, COUNT, DELETE, DISPLAY, EDIT, EXPORT, JOIN, SCAN, LABEL, LIST, LOCATE, RECALL, REPLACE, REPORT, SORT, SET FILTER, SUM, TOTAL.

Каждая из указанных команд (кроме SET FILTER) имеет опцию NOOPTIMIZE, которая при необходимости оптимизацию отключает. Такая потребность возникает довольно редко, например когда при исполнении команды может быть изменено поле, являющееся компонентом индексного выражения, и то только когда это может повлечь неполную или неправильную обработку данных (см. предупреждение выше).

Следующей командой можно полностью отключить оптимизацию:

■ SET OPTIMIZE ON/OFF

По умолчанию SET OPTIMIZE ON. Для оптимизации программисту не нужно выполнять никаких специальных действий — она будет автоматически осуществляться при наличии соответствующего открытого индексного файла. Однако желательно, чтобы при этом ни один индекс не был главным. Это значит, что они должны быть открыты командой USE или SET INDEX TO с опцией ORDER 0 или использована команда SET ORDER TO 0. Такая база будет иметь активный индекс, но не будет упорядочена. Фактором, также влияющим на эффективность оптимизации, является состояние команды SET DELETED. Желательно перед выполнением команды, использующей технологию Rushmore, установить ее параметр в OFF.

При оптимизации, естественно, требуются некоторые дополнительные ресурсы памяти компьютера. Для работы с большими базами (сотни тысяч записей) ее может не хватить. В этом случае выводится сообщение "Not enough memory for optimization" ("Недостаточно памяти для оптимизации"). При этом данные не будут потеряны и обработка продолжится, но уже обычным образом.

Результат оптимизации поиска полностью определяется оптимизируемым выражением (FOR-условием). Такое выражение называется простым, если оно содержит только одно индексное выражение, связанное знаком отношения (<, >, =, <=, >=, <>, #) с переменными, константами и другими полями несвязанных баз данных.

Пусть база KADR.DBF проиндексирована по полям POL, MONTH(DTR) и SZAR. Тогда выражения POL = 'M', MONTH(DTR)=5 и 3000 >= SZAR будут оптимизируемыми.

Простые оптимизируемые выражения могут быть связаны друг с другом и с неоптимизируемыми выражениями с помощью логических операторов AND, OR, NOT. Итоговое выражение будет полностью оптимизируемым, если все его операнды полностью оптимизируемы. В других случаях результат определяется следующим образом:

—Выражение 1—		—Выражение 2—	—Результат—
Оптимизируемое	AND	Неоптимизируемое	Частично оптимизируемое
Оптимизируемое	OR	Неоптимизируемое	Неоптимизируемое

П р и м е р:

```
FOR .NOT.pol='M'           - полностью оптимизируемое
FOR 5000>=szar.AND.pol='Ж' - полностью оптимизируемое
FOR MONTH(dtr)=7.AND.sem='Б' - частично оптимизируемое
FOR MONTH(dtr)=7.OR.sem='Б' - неоптимизируемое
```

Следующая команда поиска LOCATE будет работать подобно команде ускоренного поиска SEEK при просмотре базы KADR.DBF на соответствие условию 6000 < szar.AND.szar < 10000, что, несмотря на наличие в ключе поиска и неоптимизируемого выражения DET#0, даст экономию времени:

```
USE kadr INDEX kadrskar ORDER 0
LOCATE FOR 6000<szar.AND.szar<10000.AND.det#0
```

Таким образом, в отличие от команды SEEK команда LOCATE дает возможность задать сложное условие. Вместе с тем, если это не требуется, лучше использовать команду SEEK. С помощью круглых скобок можно строить еще более сложные выражения, используя те же правила. Очень эффективно применение оптимизирующей технологии в различных командах обработки данных с FOR-условием.

Глава 10. РАБОТА С НЕСКОЛЬКИМИ БАЗАМИ

В FoxPro допускается работа сразу с многими базами данных и при этом возможно установление разнообразных связей между ними. Указатели записей в таких связанных базах будут двигаться синхронно. База, в которой указатель движется произвольно, считается старшей, а база/базы, в которой указатель следует за указателем старшей базы, — младшей. В старшей и младших базах должны быть поля, несущие какой-то общий признак, иначе, хотя связь и возможна, она будет бессмысленна. Допускается сцепление одной базы с несколькими другими. Младшие базы, в свою очередь, могут быть связаны с базами следующего уровня и т.д.

Возможно установление двух типов связей между записями двух сцепленных баз данных. Связь типа одна_запись-к-одной перемещает указатель в младшей базе таким образом, что он всегда устанавливается на первую встреченную им запись с совпадающим признаком. Остальные такие записи (если есть) остаются "не замеченными". Эта связь устанавливается просто командой SET RELATION. Связь типа одна_запись-ко-многим позволяет обратиться ко всем записям младшей базы с совпадающим признаком (команды SET RELATION и SET SKIP TO).

Оба типа связей могут быть распространены на несколько баз сразу.

Понятие о рабочих областях

В FoxPro можно обрабатывать сразу несколько файлов баз данных (до 25). Каждый такой файл типа DBF и все вспомогательные файлы (например, индексные) открываются в своей отдельной рабочей области. Переход из области в область осуществляется командой

■ SELECT <рабочая область/псевдоним>

Первые десять рабочих областей идентифицируются номерами 1 — 10 или буквами A — J. Области с 11-й по 25-ю обозначаются номерами или буквенно-цифровыми именами W11 — W25. Если в качестве параметра указать цифру 0, произойдет переход в первую свободную рабочую область. Кроме того, рабочие области и файлы базы данных могут идентифицироваться так называемыми псевдонимами. Псевдонимом области по умолчанию является само имя находящегося в ней файла базы данных. В качестве псевдонима можно указать и любое другое слово в команде USE. Использование псевдонима позволяет при работе с разными базами называть их одним именем (псевдонимом). Это делает программу независимой от имени конкретной базы.

Область, в которой мы находимся в данный момент, называется активной рабочей областью, и в ней можно работать с находящейся здесь базой данных, используя все допустимые команды системы. Одновременно даже в одной команде можно иметь доступ (с некоторыми ограничениями) к полям других баз. В этом случае имя поля из неактивной области — составное. Собственно имени поля тогда предшествует имя рабочей области или псевдоним, разделенные знаками "-" и ">" или (что более удобно) точкой:

<рабочая область/псевдоним> -> <имя поля>

или <рабочая область/псевдоним>.<имя поля>

Например, следующие составные имена для поля FAM из базы KADR.DBF идентичны, если эта база открыта в области A или 1:

KADR.FAM, A.FAM, KADR->FAM, A->FAM.

Рабочая область в составном имени указывается любым разрешенным образом (буквой, именем ее базы данных, псевдонимом), но не номером. Номер может быть указан в команде SELECT и в функциях. В каждой области поддерживаются свои указатели записей, на положение которых не влияют переходы между областями.

При входе в СУБД активизируется область 1 (или A), и, если вы работаете только с одной базой, заботиться об открытии областей не нужно.

Пр и м е р. В файле KADR среди прочих содержатся сведения о фамилиях и табельных номерах работников, а в файле BRIG1 — о табельных номерах TAB и выработке VIR. Необходимо по фамилии (например, МИРОНОВ) из файла KADR найти его выработку из файла BRIG1.

```
.SELECT a           - переход в область A
.USE brig1         - открытие файла BRIG1.DBF
.SELECT b           - переход в область B
.USE kadr           - открытие файла KADR.DBF
.LOCATE FOR fam='МИРОНОВ' - поиск записи с фамилией МИРОНОВ
.SELECT a           - переход в область A
.LOCATE FOR tab=b.tab - поиск записи в файле BRIG1 с табельным номером
                     МИРОНОВА
.? b.fam,          tab, vir - выдача данных из обеих баз
МИРОНОВ Р.И. 468 204.00
```

В команде USE можно одновременно указывать и область, в которой открывается база. Однако переход в указанную область здесь не происходит. Так, после выполнения команд USE brig1 IN a и USE kadr IN b мы останемся в текущей области A и команда SELECT B понадобится все равно.

Очевидно, что такой способ связи файлов очень трудоемок, если требуется установить связь не с одной, а с несколькими фамилиями. В FoxPro имеется команда, упрощающая эту задачу.

Связь вида одна_запись-с-одной. Команда

■ SET RELATION TO <ключ> INTO <область>
[,<ключ> INTO <область>...] [ADDITIVE]

связывает указатель записей в активной рабочей области с указателями записей из других рабочих областей, имена которых указаны после слова INTO, по заданному общему полю (ключу). Единственное условие — файл, с которым устанавливается связь, должен быть проиндексирован по этому полю.

Пр и м е р. Связать файлы KADR.DBF и BRIG1.DBF по полю TAB. Вывести для каждого табельного номера файла BRIG1.DBF соответствующую фамилию и выработку.

```
.USE brig1 IN a
.USE kadr INDEX kadrtab IN b
.SET RELATION TO tab INTO b
.LIST tab,vir,b.fam
Record #  TAB      VIR  B.FAM
        1   98   446.00  ПОТАПОВ Д.П.
        2    6   480.72  КУЛАКОВА М.И.
        3   13   12.00  СИДОРОВ П.С.
        4  468   204.00  МИРОНОВ Р.И.
```

Здесь выведены записи файла BRIG1.DBF, в которые включено поле соответствующих им фамилий KADR.DBF (B.FAM).

В FoxPro имеется возможность устанавливать связь с несколькими базами одновременно. Если со старшим файлом, который уже связан с другим, необходимо связать некоторый третий (четвертый и т.д.), следует во все последующие команды SET RELATION включить слово ADDITIVE, которое обеспечит сохранение связей, установленных ранее.

Связь между всеми файлами разрывается командой SET RELATION TO без параметров. Связь с отдельным файлом в заданной <области> — командой

■ SET RELATION OF INTO <область>

Связь вида одна_запись-со-многими. Следующая команда устанавливает связь такого типа между двумя или несколькими базами данных:

■ SET SKIP TO [<область1> [<область2>] ...]

При этом с каждой записью из старшей базы могут быть сцеплены несколько записей из младшей базы. Связь может быть установлена сразу с несколькими младшими базами, находящимися в указанных <областях>.

Прежде чем использовать команду SET SKIP TO, необходимо выполнить начальное сцепление вида одна_запись-с-одной командой SET RELATION. Удаление связи одна_запись-со-многими осуществляется командой SET SKIP TO без параметров.

П р и м е р. Рассмотрим пример организации связи одной базы с несколькими базами, причем эта связь будет реализована по схеме одна_запись-со-многими.

Положим, что в бригадных файлах BRIG3.DBF и BRIG5.DBF некоторые табельные номера могут встречаться несколько раз (например, если фиксируются выработки каждого работника по отдельным нарядам). Допускается также, что рабочие могут работать сразу в нескольких бригадах. Требуется для каждой фамилии и табельного номера из базы KADR.DBF предъявить все выработки данного работника из баз BRIG3.DBF и BRIG5.DBF, которые проиндексированы по полю TAB (индексы BRIG3.IDX и BRIG5.IDX).

BROWSE-окно показано на рис.10.1, решение приведено ниже:

```
.SELECT a
.USE kadr IN a                                && Открытие старшей базы в области A
.USE brig3 IN b INDEX brig3                  && Открытие младшей базы в области B
.USE brig5 IN c INDEX brig5                  && Открытие другой младшей базы в C
*Установление связи одна-с-одной KADR.DBF с BRIG3.DBF, BRIG5.DBF
.SET RELATION TO tab INTO b, tab INTO c
*Установление связи одна-со-многими базы KADR.DBF с
.SET SKIP TO b,c                             && базами BRIG3.DBF и BRIG5.DBF
.BROWSE FIELDS a.fam :H='Фамилия', a.tab :H='Табель', ;
        b.tab :H='Таб/Бриг3', b.vir :H='Выр/Бриг3', ;
        c.tab :H='Таб/Бриг5', c.vir :H='Выр/Бриг5'
.SET RELATION TO                             && Отмена всех связей
```

Фамилия	Табель	Таб/Бриг3	Выр/Бриг3	Таб/Бриг5	Выр/Бриг5
ЕФИМОВ А.П.	446	446	280.50	446	50.00
		446	130.00	446	80.65
				446	100.20
ЛАРИОНОВ Т.С.	321	321	25.70	321	650.00
		321	60.00		

Рис.10.1

Для наглядности здесь выведены табельные номера работников из всех бригад. Видим, что ЕФИМОВ А.П. с табельным номером 446 в файле BRIG3.DBF имеет две записи, а в файле BRIG5.DBF — три, ЛАРИОНОВ Т.С. в бригаде номер 3 — две записи, а в бригаде номер 5 — одну. Здесь повторяющиеся поля из старшей базы отображены символом заполнения "■". Для быстрого перемещения от записи к записи в старшей базе можно использовать клавиши Ctrl-<вниз>/<вверх>.

Хотя клавиши дополнения и удаления (Ctrl-N, Ctrl-T) здесь доступны, они действуют только на старшую базу. Если при этом нужно, чтобы что-то происходило и с младшими базами, следует их перепрограммировать.

Команды DISPLAY/LIST предъявляют записи похожим образом, но для каждой сцепленной записи из младших баз значения полей из старшей базы будут повторяться. Ниже приведен результат выполнения команды

.LIST a.fam,a.tab, b.tab,b.vir,c.tab,c.vir OFF :

A.FAM	A.TAB	B.TAB	B.VIR	C.TAB	C.VIR
ЕФИМОВ А.П.	446	446	280.50	446	50.00
ЕФИМОВ А.П.	446	446	130.00	446	80.00
ЕФИМОВ А.П.	446	446	0.00	446	100.00
ЛАРИОНОВ Т.С.	321	321	25.70	321	650.00
ЛАРИОНОВ Т.С.	321	321	60.00	321	0.00

Рассмотренный пример соответствует сцеплению одной базы с несколькими. При этом реализовано два уровня данных. База KADR.DBF образует старший, первый уровень, а базы BRIG3.DBF и BRIG5.DBF — младший, второй уровень (рис.10.2).

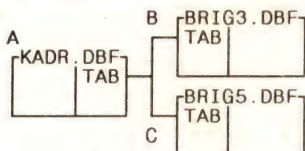


Рис.10.2

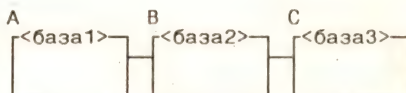


Рис.10.3

Сцепление баз можно распространить и на большее число уровней. Например, пусть имеется три базы данных (<база1>, <база2>, <база3>), в которых содержатся данные о предприятиях, цехах для всех предприятий и участках для всех цехов всех предприятий (рис.10.3). Необходимо связать эти базы таким образом, чтобы для каждой текущей записи из базы предприятий указатель записей базы цехов становился на первую запись, где находятся сведения о цехе, принадлежащем данному предприятию, а указатель в базе участков — на запись с информацией об участке этого цеха. Естественно, что все базы должны иметь поля, позволяющие осуществить их содержательное сцепление: база цехов должна иметь поле с названиями предприятий, а база участков — поле с названиями предприятий и поле с номерами цехов. Именно по этим полям должно быть сделано индексирование.

В отличие от предыдущего примера, где сцепление базы KADR.DBF выполнялось со всеми подчиненными базами сразу, здесь команды SET RELATION TO вводятся по мере перемещения из области в область. Схема организации связей приведена ниже:

```
USE <база1> IN a
USE <база2> IN b INDEX <индекс2>
USE <база3> IN c INDEX <индекс3>
SELECT a
SET RELATION TO <ключ базы1> INTO b && Сцепление Базы1 с Базой2
SELECT b
SET RELATION TO <ключ базы2> INTO c && Сцепление Базы2 с Базой3
```

Разбор конкретного примера пока отложим (см. разд.22.2).

Аппарат сцепления баз командой SET RELATION является мощным средством доступа к "родственным" данным. Однако, поскольку такое сцепление влечет синхронное перемещение указателей записей во всех подчиненных базах вслед за перемещением указателя в главной базе, это может отнимать много времени в случае, если доступ к младшей базе в данный момент не нужен. Поэтому часто бывает целесообразным временное разъединение баз. Во многих случаях вообще лучше прибегнуть к поиску нужной записи командой/функцией SEEK, нежели к установлению постоянной связи.

Глава 11. СОЗДАНИЕ КОМАНДНЫХ ФАЙЛОВ

К настоящему моменту нами были рассмотрены некоторые основные команды языка FoxPro. Их практическое освоение, как мы надеемся, дало вам возможность почувствовать мощь инструмента FoxPro и власть над данными. Однако для построения полноценных систем обработки данных этого, конечно, недостаточно. Очевидно, что не хватает пока средств управления вычислительным процессом, организации диалога, структурирования программ, не обсуждался вопрос о временных переменных и массивах и т.д. Кроме того, во многих случаях такие команды оказываются слишком "грубыми" для работы в прикладных системах обработки данных.

Дальнейший материал посвящен изучению команд, ориентированных на прямое программирование процессов обработки данных. Такие программы создаются с помощью внутреннего встроенного текстового редактора FoxPro, вызываемого командой

- **MODIFY COMMAND/FILE** <имя текстового файла/маска>
[NOEDIT] [NOWAIT] [WINDOW <окно>]

Если используется вариант **MODIFY COMMAND**, по умолчанию вновь создаваемый файл считается командным (программным) файлом и ему присваивается расширение PRG, если оно не задано явно. Повторный вызов командного файла осуществляется этой же командой. Если вы забыли, как называется ваша программа или хотите просмотреть сразу несколько файлов, можно задать <маску> с использованием обычных замещающих символов DOS — "*" и "?". Например, команда **MODIFY COMMAND F*** вызовет на экран все файлы типа PRG, имена которых начинаются на букву F, каждое в свое окно, а команда **MODIFY COMMAND *** — вообще все PRG-файлы. Далее нужные окна можно оставить на редактирование, а остальные убрать, нажав клавишу **Escape** или установив маркер мыши в левый верхний угол, помеченный прямоугольником, и нажав кнопку мыши. Вариант **MODIFY FILE** не предусматривает каких-либо умолчаний для имен файлов.

Внутренний редактор FoxPro применяется не только для работы с программами. Именно он используется для предъявления мемо-полей при нажатии клавиш **Ctrl-Home**. Кроме того, его удобно привлекать для отображения текстовых файлов внутри прикладной системы обработки данных, например для того, чтобы пользователь мог решить, печатать данный документ или нет. По умолчанию редактор использует собственное системное окно, в заголовок которого выносятся имя файла. Однако его можно "посадить" в любое предварительно определенное командой **DEFINE WINDOW** окно с нужным заголовком и заданным местонахождением. Для этого в команду следует включить опцию **WINDOW <окно>**. Нежелательный доступ пользователя к данным при этом может быть заблокирован опцией **NOEDIT**. Окно редактора может присутствовать на экране совместно с другими окнами, меню и прочими объектами прикладной системы. Чтобы открывающееся окно редактора не препятствовало активации других объектов текущего экрана, в команду включается опция **NOWAIT**.

Действие команды **MODIFY COMMAND** в общем не отличается от действия команды **MODIFY FILE**, за исключением одной особенности. Первая команда выводит строки, не "ломаая" их у правого края окна. Это может быть удобно в случае, если выводимый в прикладной системе текст имеет размер строки больший, чем используемое окно. Очень часто в программах генерируются тексты (например, таблицы), имеющие больше 80 колонок. Доступ к данным, оказавшимся за пределами окна, в этом случае может быть осуществлен

клавишами управления курсором. Команда же **MODIFY FILE** автоматически разбивает длинную строку на несколько коротких, чтобы уместить ее всю на экране, превращая таким образом в "кашу" выведенную таблицу. Разбиение строк может быть предотвращено с помощью меню настройки редактора с последующим запоминанием ее в файле **FOXUSER.DBF**. При этом прикладная система для эксплуатации должна иметь доступ к этому файлу. Если нет других причин использовать в программе файл **FOXUSER.DBF**, лучше применять команду **MODIFY COMMAND**.

Созданная в редакторе программа запоминается в указанном в команде файле и может быть в дальнейшем вызвана на исполнение командой

■ DO <имя командного файла>

Расширение **PRG** указывать не обязательно.

Исполнение программы может быть прервано в любой момент нажатием клавиши **Escape**, если командой

■ SET ESCAPE ON/OFF

установлено **ON** (действует по умолчанию). В готовой программе эта возможность должна быть подавлена параметром **OFF**.

Рассмотрим возможности встроенного редактора **FoxPro**.

В редакторе действие традиционных клавиш управления (перемещения курсора, **PgDn**, **PgUp**, **Del**, **Ins** и др.) имеет обычный смысл. Кроме того, есть возможность быстрого перемещения курсора клавишами:

Ctrl-<вправо/влево>	— на слово вправо/влево,
Home/End	— на начало/конец строки,
Ctrl-Home/End	— на начало/конец текста.

После редактирования программы сохранение ее текста в файле осуществляется клавишами **Ctrl-W**. Выход нажатием клавиш **Escape** или **Ctrl-Q** не влечет перезаписи текста в файл, однако, если перед этим было сделано изменение данных, сначала появится меню-предупреждение ("Discard changes?") о возможной необходимости сохранить данные.

В редакторе можно удалять/перемещать целые фрагменты текста. Выделение текста для манипуляций с ним осуществляется при одновременном нажатии клавиши **Shift** и одной из клавиш управления курсором. При этом выделяются следующие фрагменты текста:

Shift-<вправо/влево>	— символа справа/слева,
Shift-<вверх/вниз>	— строки,
Shift-Ctrl-<вправо/влево>	— до конца/начала слова,
Shift-Ctrl-End/Home	— до конца/начала текста,
Ctrl-A	— всего текста.

Выделение очень удобно выполнять мышью. Для этого надо поставить ее маркер на начало выделяемого участка, нажать кнопку мыши и, не отпуская ее, переместить курсор в конец участка. Кроме того, двойное/тройное нажатие кнопки дает возможность сделать быстрое выделение слова/строки.

Снятие выделения выполняется нажатием любой клавиши или перемещением мыши. Выделенный фрагмент может быть взят в буфер клавиатуры с удалением (клавишами **Ctrl-X**) или без удаления (**Ctrl-C**) его из текста. Текст, взятый в буфер, может быть извлечен из него в любое другое место, в том числе и в другом файле (в окне, открытом другой командой **MODIFY COMMAND**) с помощью клавиш **Ctrl-V**. Удаление выделенного фрагмента текста без взятия его в буфер осуществляется нажатием клавиш **Delete/Backspace**.

Пусть требуется перенести фрагмент текста из одного файла в другой. Для этого сначала нужно поставить курсор в начало выделяемого участка в исходном файле и при нажатой клавише **Shift** выделить (иным цветом)

нужный фрагмент клавишами со стрелками, PgDn/PgUp или другими клавишами или мышью. Затем следует взять этот текст в буфер нажатием клавиш Ctrl-X (с удалением из исходного текста) или Ctrl-C (без удаления). Далее следует загрузить файл, в который переносится текст, установить курсор в нужное место и извлечь из буфера на экран содержащийся там текст нажатием клавиш Ctrl-V.

Фрагменты текста можно извлекать с помощью клавиш Ctrl-C и из окна Help. Особенно удобно брать из него готовые примеры и пробовать их в своих командных файлах.

Удаление слова, в котором находится курсор, выполняется клавишами Ctrl-Backspace. Если курсор стоит в промежутке между словами, то удаляется слово, стоящее справа. Удаление других фрагментов текста выполняется выделением с последующим нажатием клавиш Delete/Backspace или Ctrl-X.

Редактор поддерживает режим отката. Если вы хотите отказаться от последней операции, сделанной в редакторе, то следует нажать Ctrl-U. Если все-таки это действие оказалось необходимым, можно вернуться в исходное положение, нажав Ctrl-R.

В редакторе возможны поиск вхождения (Ctrl-F и Ctrl-G), замена вхождения (Ctrl-E), а также некоторые другие действия.

Полное меню редактора можно увидеть, выбрав позицию Edit в главном меню FoxPro. Возможна настройка редактора в пункте Preferences меню Edit. Настройки перечислены в гл. 28, где описывается системное меню FoxPro.

Если вас не устраивает внутренний редактор FoxPro, можно воспользоваться любым внешним редактором, к которому вы привыкли, указав его имя в файле CONFIG.FP.

Комментарии. Для ориентирования в текстах программ необходимы комментарии. Отдельная строка комментариев должна начинаться со звездочки (*). Комментарии могут быть внесены и в строку, где уже есть команда. Для этого они предваряются двумя знаками &&. Комментарии не влияют на обработку данных. В случае, если требуется временно исключить из программы значительный фрагмент, можно ограничить его операторами IF...ENDIF, указав в качестве условия "Ложь" (IF .F.).

Создадим очень простую систему обработки данных — контроль исполнения документов (КИД). Такие системы разной степени сложности обычно включают в АРМы руководителя. Программа, которую мы разберем, является довольно удобной, несмотря на то, что реализована она по существу одной командой BROWSE. Поставим задачу. Пусть требуется хранить сведения об исполняемых документах в порядке намеченных сроков исполнения, а также выделить каким-то образом просроченные документы и документы, срок исполнения которых сегодня и завтра.

Запись такой базы данных (назовем ее KID.DBF) имеет структуру, представленную на рис.11.1.

Содержание	Название	Тип	Длина
Дата исполнения	DAT	Дата	8
Номер документа	NDOK	Число	5
Содержание документа	SOD	Строка	60
Исполнители	ISP	Строка	60

Рис.11.1

Для ее создания воспользуемся командой CREATE kid .

Задача ранжирования базы по дате легко решается индексированием ее по полю DAT. Назовем этот индексный файл KID.IDX.

.INDEX ON dat TO kid COMPACT

Программа приведена ниже.

```
*----- KID.PRГ - контроль исполнения -----
SET TALK OFF
SET DATE GERMAN
SET DELETE ON
CLEAR
d=DATE()
USE kid INDEX kid
BROWSE TITLE 'Сохр.-~W      Вых.-Esc      Доп.-~N      Удал.-~T'+;
      '(сегодня '+DTOC(DATE()))' COLOR SCHEME 10;
FIELD stat=IIF(dat<d,'* Проср.', IIF(dat=d,'Сегодня',;
      IIF(dat=d+1,'Завтра',))) :H='Статус' :8,;
      ndok :H='Номер' ;;
      dat :H='Дата' ;;
      sod :H='Содержание' :27,;
      isp :H='Исполнители' :24
PACK
USE      &&-----Конец модуля-----
```

В примере сначала подавляются ненужные в данном случае автоматические сообщения FoxPro о результатах выполнения команд (SET TALK OFF) и устанавливается удобный формат даты (SET DATE GERMAN). Чтобы сделать невидимыми записи, помеченные к удалению в процессе работы с документами, используется команда SET DELETE ON. Экран очищается (CLEAR). Далее запоминается текущая дата в переменной D (d=DATE()) и открывается база данных KID.DBF совместно с индексным файлом KID.IDX (USE kid INDEX kid).

Теперь главное — определение облика и функций BROWSE-окна. Цвет зададим стандартной цветовой схемой 10 для BROWSE-окон (COLOR SCHEME 10). В заголовке (TITLE) укажем функции стандартных клавиш и текущую дату. Для того чтобы дата сочеталась с остальной информацией строкового типа в заголовке, она приводится к этому типу функцией DTOC(DATE()).

Все поля базы данных получают соответствующие заголовки. Видимая часть полей ISP и SOD ограничивается 27 и 24 символами, поскольку целиком (120 символов) они не могут быть отображены на экране. Доступ к скрытой части полей возможен с помощью клавиш управления курсором.

Вид экрана, предьявляемого программой, изображен на рис.11.2.

Сохр.-~W Статус	Вых.-Esc Номер	Доп.-~N Дата	Удал.-~T Содержание	(сегодня 22.11.92) Исполнители
* Проср.	10	02.11.92	Профсоюзное собрание	Профком, Прокофье
Сегодня	24	22.11.92	Выделение работников	Отдел кадров, Нач
Завтра	27	23.11.92	0 распределении фонд	Председатель СТК,
	117	06.12.92	0 готовности финансо	Гл. бухгалтер, На
...

Рис.11.2

Запись базы дополнена вычисляемым полем STAT, в котором отображается статус документа (Просрочен, Исполнение сегодня, Исполнение завтра) в зависимости от того, меньше дата исполнения документа текущей даты (dat<d), равна или больше на единицу. Просроченные документы выделяются также звездочкой.

Глава 12. КОМАНДЫ ВВОДА-ВЫВОДА

Если стандартная форма окна редактирования вам кажется неудобной, можно прибегнуть к индивидуальному определению места и формы предъявления данных для каждого поля/переменной, используя специальные команды ввода/редактирования. Но сначала рассмотрим команды очистки экрана.

■ CLEAR

Команда освобождает весь экран/окно от имеющейся информации и устанавливает курсор в левый верхний угол экрана/окна.

■ @ <Y1,X1> [CLEAR/CLEAR TO <Y2,X2>]

Команда очищает в окне/экране прямоугольник с координатами верхнего левого угла Y1, X1 и нижним правым углом окна/экрана (если указана опция CLEAR) или произвольными координатами Y2, X2 нижнего правого угла (если указана опция CLEAR TO ...).

Пр и м е р. Очищаются прямоугольники.

```
.@ 5,0 CLEAR      && Координаты: 5,0 и правый нижний угол
.@ 15,5 CLEAR TO 20,50  && Координаты: 15,5 на 20,50
```

Команда ввода-вывода @...SAY...GET

Команда является наиболее универсальной командой такого типа. Она используется для форматированного ввода-вывода данных на экран/принтер.

■ @ <Y,X>

```
[SAY <выр1>
 [PICTURE <вырC1>] [FUNCTION <Фкоды1>]
 [COLOR SCHEME <вырN1>/COLOR <список цветовых пар>]]
[GET <пер>
 [PICTURE <вырC2>] [FUNCTION <Фкоды2>]
 [DEFAULT <выр2>] [ENABLE/DISABLE]
 [MESSAGE <вырC3>] [[OPEN] WINDOW <окно>]
 [RANGE [<выр3>] [,<выр4>]]
 [SIZE <вырN2>,<вырN3>]
 [VALID <вырL1>/<вырN4>] [ERROR <вырC4>]]
 [WHEN <вырL2>]
 [COLOR SCHEME <вырN5>/COLOR <список цветовых пар>]]
```

Здесь Y и X — пара чисел или переменных, которые определяют номер строки и столбца (для экрана это соответственно 0 — 24 и 0 — 79, для принтера определяется размером листа бумаги). Именно с этой позиции будет осуществляться ввод-вывод. Обязательным элементом команды является только @ <Y,X>. Если больше ничего нет, то курсор устанавливается в позицию экрана Y, X и очищает строку Y вправо с позиции X.

Параметры команды:

@ Y,X SAY <выр1> — Выдает с заданной позиции <выражение> любого типа данных (поля базы данных, переменные, элементы массивов). Разрешается указывать сложное <выражение>, состоящее из нескольких элементов, соединенных знаком сцепления "+". Все такие элементы тогда должны быть символьного типа или приведены к нему, например, функцией STR(). В качестве <выражения> может быть использована ПФ. Пусть числовая переменная X=7. Тогда команда

```
@ 4,8 SAY 'HOME' - '+'STR(X,1)
```


выдаст строку НОМЕР - 7.

@ Y,X GET <переменная/поле> — с заданной позиции выводится переменная или поле записи с возможностью их редактирования. Переменная до этого должна существовать (если не указана опция DEFAULT).

DEFAULT <выр2> — опция действует только при работе с переменными и не влияет на поля баз данных. Она задает выражение, которое по умолчанию будет помещено в GET-переменную и предъявлено на редактирование. Опция может создавать временные переменные и присваивать им исходные значения. Однако если переменная ранее существовала и имела какое-то значение, то именно это значение и будет предъявлено на редактирование. Если вы хотите гарантированно задать исходное значение в опции DEFAULT, можно, например, предварительно уничтожить переменные командой RELEASE.

ENABLE/DISABLE — разрешается/запрещается доступ к GET-полю. Цвета таких полей определяются соответственно шестой/десятой цветовыми парами из цветовой схемы номер 1.

MESSAGE <вырC3> — для данного поля задает поясняющее сообщение <вырC3>, выводимое в последней строке экрана/окна (если не изменено командой SET MESSAGE).

RANGE <выр3>,<выр4> — этот элемент команды организует входной диапазонный контроль вводимых величин любого типа данных (числовые, строковые, даты). Здесь проверяется вхождение редактируемой переменной в заданный диапазон от <выр3> до <выр4>. Допускается отсутствие одной из границ — <выр3> или <выр4>. Если значение введенной переменной не находится внутри заданного диапазона, раздается сигнал и появляется сообщение о допустимых границах

RANGE: <выр3> to <выр4>

Для указания собственного сообщения может быть использована команда ON READERROR, перехватывающая сообщения об ошибках ввода. Нажатие клавиши Enter без изменения <переменной> не влечет проверки соответствия границам.

SIZE <вырN2>,<вырN3> — определяет область, отводимую под редактируемое поле. По умолчанию под редактируемое поле/переменную отводится одна строка длиной, равной длине поля/переменной. Если она не умещается на экране/окне, то строка "ломается" у правой границы и продолжается на следующей строке и т.д. Опция SIZE позволяет управлять этим процессом. Эдесь <вырN2> — число строк, а <вырN3> — число колонок, отводимых под данные. Если фактическая длина данных больше обозначенной области, доступ к неуместившейся их части может быть осуществлен перемещением курсора (скролингом).

П р и м е р (результат показан справа):

```
f='Петропавловский А.
@ 2,4 SAY 'Фамилия:' GET f SIZE 3,6
READ
```

```
Фамилия: Петроп
авловс
кий А.
```

Команда READ осуществляет собственно считывание данных из редактируемого поля.

VALID <вырL1>/<вырN4> [ERROR <вырC4>] — если контроль должен быть более сложным, можно предусмотреть логическую проверку вводимой переменной в фразе VALID. Если указано условие <вырL1>, то будет допущен ввод только такого значения, которое ему удовлетворяет, т.е. <вырL1>=.Т.. Если введено неправильное значение (<вырL1>=.F.),

система выдаст предупреждение о неправильном вводе

Invalid Input

и предложит после нажатия клавиши Пробел повторить ввод. Можно указать собственное сообщение <вырС4> на неправильный ввод, используя параметр ERROR. Опция VALID (в отличие от RANGE) выполняет проверку всегда и допустит выход из редактируемой области только при нажатии клавиши Escape, если результаты проверки оказались неверными (даже если редактирование не выполнялось). Допускается применять пользовательскую функцию, которая должна возвращать значение логического или числового типа. Если в опции VALID получено числовое выражение <вырN4>, оно определит относительное положение поля, которое будет редактироваться после текущего. Выражение может быть положительным (движение вперед) и отрицательным (движение назад). Если <вырN3> указывает на номер поля, которое отсутствует, команда прерывается. Если <вырN>=0, значит, обнаружена ошибка ввода и курсор остается в том же поле. Никаких сообщений об ошибке при этом не выводится. В сложном случае и проверка ввода, и возможные сообщения могут быть реализованы в пользовательской функции, вызываемой предложением VALID.

WHEN <вырL2> — вход в редактируемое поле допускается только при истинности условия <вырL2>. Если условие имеет значение .F., поле пропускается.

[OPEN] WINDOW <окно> — опция используется с мемо-полями. Редактируемое мемо-поле открывается в определенном ранее командой DEFINE WINDOW окне. Если указано слово OPEN, окно будет открыто по умолчанию. Во всех случаях для входа в мемо-поле на экране необходимо нажать Ctrl-Home или другие клавиши доступа к мемо-полю. Более удобно здесь использовать команду @...EDIT.

COLOR SCHEME <вырN>/COLOR <список цветовых пар> — определяет раскраску областей ввода-вывода. По умолчанию используется цветовая схема номер 1: первая цветовая пара для SAY-вывода, вторая — для GET-областей.

Возможно соединение фраз SAY и GET в одной команде. Тогда область ввода GET предъявляется непосредственно после сообщения SAY в той же строке.

Команды @...GET позволяют осуществить только предъявление данных. Наделение GET-полей аппаратом редактирования и фактическое запоминание экранных образов данных в полях/переменных осуществляются другой командой:

■ READ

которая стоит обычно всегда вслед за командой/командами @...GET.

В FoxPro-2.0 возможности команды READ значительно расширены — она получила ряд опций по управлению вводом. Теперь с помощью одной команды READ можно осуществлять ввод данных в нескольких независимых окнах, а также иметь доступ к световым меню, окнам BROWSE и окнам, открытым командой MODIFY FILE/COMMAND/MEMO.

Такие средства команды READ как раз и дают возможность создать интерфейс, "управляемый событиями". Значение этой команды трудно переоценить. Однако сейчас ограничимся только ее простейшей формой. Более полный формат будет рассмотрен в гл.26, а пока дополнительные возможности при необходимости разъясняются прямо в тексте.

Пр и м е р. Пусть требуется осуществить ввод некоторого платежа за купленную продукцию в переменную PL, значение которой может находиться в диапазоне 100 — 60 000 руб. ввиду того, например, что банк не принимает суммы менее 100 руб., а предприятие не может выплатить более 60 000 руб. сразу. При вводе необходимо предусмотреть выдачу подсказки (слово ПЛАТЕЖ) и контроль заданного диапазона:

```
pl=100
@ 10,8 SAY 'ПЛАТЕЖ -' GET pl RANGE 100,60000
READ
```

При выполнении данного фрагмента программы в десятой строке с восьмой колонки экрана появится слово ПЛАТЕЖ, непосредственно за которым контрастным цветом обозначится редактируемое значение, равное 100:

ПЛАТЕЖ - 100

Курсор установится на первой позиции редактируемой области. Затем можно ввести новое значение PL или оставить старое. После завершения редактирования следует нажать клавишу Enter. Если при этом была сделана попытка ввода числа, находящегося вне заданного диапазона, или нецифрового символа, последует соответствующее сообщение и после нажатия клавиши Space ввод будет нужно повторить.

Усложним пример. Пусть также известно, что единица приобретаемой продукции стоит 450 руб. и поставщик предлагает ее партиями не менее четырех и не более 50 штук, т.е. сумма должна иметь минимальное значение $4 \cdot 450$, максимальное $50 \cdot 450$ и быть кратной 450. На языке математики это означает, что остаток от деления на 450 вводимого числа должен быть равен нулю. Таким образом, команда ввода получит следующие ограничения:

```
RANGE 4*450,50*450 и VALID MOD(pl/450)=0
```

Ввиду того что условия поставки могут меняться, лучше минимальное и максимальное количество и цену использовать в виде переменных (например, MX, MN, CEN), значения которых должны быть заданы ранее. Окончательно команда будет выглядеть так:

```
@ 10,8 SAY 'ПЛАТЕЖ -' GET pl VALID MOD(pl/cen)=0;
RANGE MIN(100,MN*CEN), MAX(60000,MX*CEN)
```

Здесь функция MOD() вычисляет остаток от деления PL на CEN, а функции MIN()/MAX() — минимальное/максимальное от ограничений покупателя и поставщика.

Вернемся к разбору команды @...SAY...GET. При необходимости можно контролировать не только диапазон вводимых значений, но и их форму, используя фразу PICTURE <шаблон> и/или FUNCTION <Фкоды>.

Шаблон состоит из специальных символов, которые устанавливает программист в соответствии с желаемой формой вводимой/выводимой переменной. Каждый символ шаблона определяет один символ переменной. Строка шаблона может содержать любые знаки, однако только символы, которые будут перечислены ниже (и символы "\$", ",", действие которых для отечественного пользователя бесполезно), влияют на данные. Иные символы будут лишь отображаться при вводе-выводе. При вводе курсор будет перескакивать через них. Этот механизм хорошо работает в числовых полях. В символьных полях обычно необходимо пользоваться кодом R (FUNCTION '@R...').

Символы шаблона PICTURE разрешают ввод вместо себя только определенных символов данных. В основном шаблоны предусмотрены для ввода данных (GET) и лишь иногда — для вывода (SAY). Ниже эти символы перечислены. В неочевидных случаях указано и их применение (GET, SAY):

- A — Допускает ввод только букв.
- L — Допускает ввод только логических данных вида T/F.
- N — Допускает ввод только букв и цифр.
- X — Допускает ввод любых символов.
- Y — Допускает ввод только логических данных вида Y/N.
- 9 — В символьных данных допускает ввод только цифр, в числовых — цифр и знаков "+" и "-".
- # — Позволяет вводить цифры, пробелы и знаки "+" и "-".
- ! — Преобразует строчные буквы в прописные (GET, SAY).
- * — Звездочки выводятся перед числами. Может использоваться для защиты от подделки (SAY).
- . — Точка задает позицию десятичной точки в дробном числе (GET, SAY).

Замечание к символу шаблона #. Если фактически выводимое или редактируемое число имеет больше дробных разрядов, чем предусмотрено шаблоном, лишние правые разряды отбрасываются без округления. Строка

```
@ 10,8 SAY 'ПЛАТЕЖ -' GET p1 PICTURE '#####.##'
```

допускает вместо символов # ввод знака числа и цифр с десятичной точкой, фиксированной после пятой позиции.

Аналогичные задачи решает включение в команду форматных функций ввода-вывода (FUNCTION <Фкоды>). Но коды функций (Фкоды) распространяются на всю переменную, а не на отдельные ее символы. Обычно, коды функций сочетаются с шаблонами. В этом случае нет необходимости указывать в команде само слово FUNCTION, а нужно поместить в апострофы перед строкой шаблона знак @ и коды функций и затем, обязательно через пробел — сам шаблон.

Форматные коды FUNCTION:

- A — Ввод только букв. Пробел может быть реализован перемещением курсора (GET).
- B — Выводимые числа выравниваются к левой границе поля (SAY).
- I — Выводимый текст центрируется внутри поля (SAY).
- J — Выводимый текст выравнивается к правой границе поля (SAY).
- E — Выводимые числа отображаются с ведущими нулями, а не пробелами (SAY, GET).
- K — При попадании курсора в данное поле (оно выделяется контрастным красным цветом) нажатие в начальный момент любой содержательной клавиши влечет немедленную его очистку. Если вы хотите выполнить редактирование, сначала следует нажать клавишу перемещения вправо. Режим удобен в случае радикального изменения данных (GET).
- M <список> — задает <список> вводимых элементов данных, разделенных запятыми. При вводе в поле отображается первый элемент списка. Для вызова следующего элемента списка нужно нажать клавишу Пробел или первую букву элемента. Не допускается ввод каких-либо иных данных. Функция может быть использована только с данными символьного типа (GET).
- S<n> — вывод не всей строки данных, а только ее N знаков с возможностью просмотра в этом окне остальной информации с помощью клавиатуры (GET и SAY для символьных данных).
- R -- Вывод в шаблоне символов, которые не являются частью данных (GET и SAY совместно с PICTURE). Применяется только с символьными данными. Функция R удобна для вывода символов, облегчающих восприятие информации, но которые нецелесообразно запоминать в данных. Среди этих символов не могут использоваться никакие символы шаблона (т.е. символы A L N X Y 9 # ! * . , \$).

- T — Удаляет при выводе ведущие и концевые пробелы в поле (SAY).
- Z — При выводе числа, равного нулю, выводятся все пробелы (GET, SAY).
- ! — Буквы алфавита преобразуются в прописные (GET, SAY).
- ^ — Выводит числа в экспоненциальной форме (GET, SAY).
- (— Заключает отрицательные числа в скобки (SAY).

Допускается сочетание форматных кодов между собой и со знаками шаблона. При этом символы кодов не должны противоречить друг другу.

Замечание к функциям и шаблонам A, N и !. Здесь имеются в виду только латинские символы. На русские буквы функции не распространяются. Однако имеются отечественные адаптации пакета FoxPro, где эти функции реализованы.

П р и м е р. Пусть требуется в базе данных вводить и хранить реквизиты паспортов. Как известно, номер паспорта состоит из трех римских цифр (латинских букв), черточки, двух русских букв и шести цифр (например: ХХУ-МЮ N716997).

Весь номер запоминается в символьной строке PASP длиной в 11 знаков. При вводе может быть использован следующий шаблон:

```
'@R! Серия AAA-XX номер 999999'
```

Здесь восклицательный знак конвертирует латинские буквы в прописные, буквы "А" гарантируют ввод только букв, девятки — цифр, а буквы "Х" разрешают ввод любых символов. Элементы области ввода "Серия", "-", "номер" и пробелы при вводе будут только отображаться, но запоминаться они не будут, поскольку в шаблон включен код R. Например, вышеуказанный номер будет храниться в следующем виде: ХХУМЮ716997.

В строке отсутствует какой-либо входной контроль за вводом двух русских букв, поскольку на них не действует знак "!". Если все-таки вы хотите это сделать на оригинальном пакете FoxPro, необходимо написать собственную функцию и включить ее в условие VALID. К этому вопросу мы вернемся позже. Здесь введем только самый простой способ контроля, проверив одну первую русскую букву (четвертый символ по порядку). Для этого она функцией SUBSTR() выделяется из строки ввода, а функцией BETWEEN() выясняется, находится ли введенная буква в диапазоне букв А — Я. Здесь имеется в виду, что русский алфавит представлен в альтернативной кодировке, где буквы расположены в алфавитном порядке.

```
@ 10,30 SAY 'Введите серию/номер паспорта -' GET pasp;
      PICTURE '@R! Серия AAA-XX номер 999999';
      VALID BETWEEN(SUBSTR(pasp,4,1), 'А', 'Я');
      ERROR 'Только прописными буквами' DEFAULT SPACE(11)
```

READ

Для указания пользователю на неверный ввод в опции ERROR задано предупреждение, которое будет показано в окне системных сообщений, если VALID-условие не удовлетворено.

Использование шаблонов, функций, диапазонов и условий ввода данных очень удобно для пользователя, поскольку в значительной степени защищает его от неправильного ввода информации. Это исключительно важно, так как заполнение базы данных — обычно длительный и трудоемкий процесс. Естественно, при этом желательно избежать или хотя бы уменьшить количество ошибок ввода.

П р и м е р. Шаблон ввода телефонного номера в переменную TEL, включая код города, добавочный номер (если есть) и, возможно, фамилию абонента (кого пригласить к телефону), следующий:

```
@ 2,2 GET tel PICTURE;
      '@R (999) ###-99-99 доб:##-## Спросить: !XXXXXXXXXXXX' DEFAULT SPACE(27)
```

READ

Здесь цифровые позиции, которые должны быть заполнены обязательно, имеют символ шаблона "9", а необязательные (номер телефона может иметь меньше семи цифр и без добавочного) — "#", что позволяет пропускать их при вводе. Абонент (кого спросить) обозначен символами "X" (любой ввод), кроме самого первого знака. Значок "!" обеспечивает прописную букву (для русифицированного пакета) в этом месте, поскольку здесь обычно вводится фамилия или имя. Если это должность, то хуже не будет.

Область ввода, реализованная командами, может выглядеть следующим образом (позиции ввода, оставшиеся пустыми, указаны знаком подчеркивания):

012 _ _ _ -38-67 доб: _ _ -56 Спросить: Петрова _ _ _ _ _

а содержимое переменной:

012 _ _ -3867 _ -56Петрова _ _ _ _ _

П р и м е р. Создадим формат редактирования данных в базе KADR.DBF, изображенный на рис.12.1.

ДАННЫЕ О СОТРУДНИКЕ		Сегодня 25.12.91
Фамилия, инициалы: КУЛАКОВА М.И.		
Дата рождения (день.месяц.год): 15.04.49		
Табельный номер: 6	Количество детей: 2	Пол (М или Ж): Ж
Семейное положение: Б		Ср. зарплата: 200.00
Перемещения по службе: тето		Подразделение: ОГМ
Выход с сохранением изменений - Ctrl-End,		Без - Esc
Ввод фамилии прописными буквами		

Рис.12.1

Программа, реализующая эту задачу, приведена ниже. Разъясним неочевидные элементы.

В пятой строке экрана выводится текущая системная дата (DATE()). Перемещение курсора в поле FAM сопровождается сообщением "Ввод фамилии прописными буквами" в строке MESSAGE. Вводимая дата рождения ограничена диапазоном 16 — 80 лет (трудоспособным возрастом). Поскольку пол может иметь только два значения М и Ж, они перечислены в списке альтернативных значений в функции ввода М (FUNCTION 'М М,Ж'). Здесь возможен только выбор перечисленных значений путем нажатия клавиши Spase. Аналогичным образом устроен ввод в поле SEM. Команда @ 5,9 TO 15,70 DOUBL рисует двойной линией прямоугольник, а команда @ 11,10 TO 11,69 — одинарную линию в строке 11.

```
*----- Форматный файл KADR.FMT -----*
@ 4,30 SAY 'ДАННЫЕ О СОТРУДНИКЕ'
@ 5, 9 TO 15,70 DOUBL
@ 5,51 SAY 'Сегодня '+DTCO(DATE())+' '
@ 6,18 SAY 'Фамилия, инициалы:' GET fam
MESSAGE 'Ввод фамилии прописными буквами'
@ 7,20 SAY 'Дата рождения (день.месяц.год):'
GET dtr RANGE DATE()-80*365, DATE()-16*365
@ 8,11 SAY 'Табельный номер:' GET tab
@ 8,33 SAY 'Количество детей:' GET det
@ 8,54 SAY 'Пол (М или Ж):' GET pol FUNCTION 'М М,Ж';
MESSAGE 'Выбор клавишей ПРОБЕЛ'
@ 9,15 SAY 'Семейное положение:' GET sem FUNCTION 'М Б,Х,Р';
MESSAGE 'в Браке, Холост, Разведен - выбор клавишей ПРОБЕЛ'
@ 9,42 SAY 'Ср. зарплата:' GET szar
@10,12 SAY 'Перемещения по службе:' GET per;
MESSAGE 'Нажмите Ctrl-Home'
@ 10,40 SAY 'Подразделение:' GET podr
@ 11,10 TO 11,69
@ 12,12 SAY 'Выход с сохранением изменений - Ctrl-End, Без - Esc'
```


Такую программу удобно поместить в так называемый форматный файл. Форматному файлу дается любое удобное имя, но расширение FMT. После того как он создан, непосредственно перед использованием форматный файл может быть открыт командой

■ SET FORMAT TO [<имя форматного файла>]

В форматном файле допускаются команды вида @...GET/SAY и команды CLEAR и READ.

Теперь любая команда редактирования вида EDIT/CHANGE, READ, INSERT, APPEND будет предъявлять данные на редактирование только в указанном формате. Отмена действия форматного файла осуществляется командой SET FORMAT TO без параметра. Если этого своевременно не сделать, далее все команды READ и другие команды редактирования будут пытаться к нему обращаться.

Использование любой из перечисленных команд редактирования совместно с форматным файлом сохраняет возможность перемещения внутри базы данных. Однако клавиши дополнения (Ctrl-N) и пометки к удалению (Ctrl-T) здесь не действуют.

Пр и м е р:

```
.USE kadr  
.SET FORMAT TO kadr.fmt  
.CHANGE  
.SET FORMAT TO
```

Результатом выполнения указанного примера как раз и явился изображенный рис.12.1 экран редактирования.

Форматный файл допускает многостраничный экран редактирования. Для этого страницы форматного файла следует разделить командой READ.

Форматный файл может быть использован и с командой BROWSE:

```
BROWSE FORMAT
```

При этом все SAY-сообщения интерпретируются как заголовки полей, что, очевидно, не дает возможности включать в форматный файл какие-нибудь отдельные строки. Координаты, указанные после символа @, игнорируются.

Направление информации, выдаваемой командой @ в форме @...SAY на экран, принтер или в файл, осуществляется командой

■ SET DEVICE TO SCREEN/PRINT/FILE <файл>

Форма SET DEVICE TO PRINT/FILE направляет выдачу на принтер или в <файл>, а SET DEVICE TO SCREEN — на экран. Последняя форма принята по умолчанию, т.е. существует при загрузке FoxPro в компьютер. Если необходима выдача на печать, то в соответствующее место программы включается команда SET DEVICE TO PRINT, а после завершения печати она должна быть отменена командой SET DEVICE TO SCREEN.

Данные, отображенные в GET-областях, командой игнорируются.

Привлекательность команды @...SAY...GET заключается в том, что она позволяет организовать любую форму ввода/редактирования данных в указанных диапазоне и виде. При этом соответствующая область экрана выделяется контрастным цветом. Кроме того, только эта команда и команды полноэкранного редактирования (BROWSE, CHANGE и др.) могут использоваться для непосредственного ввода данных в поля базы данных.

Если всего этого не требуется, можно использовать более простые команды ввода: INPUT, ACCEPT, WAIT.

Команда вывода ?/??

Эта команда является самой простой, но во многих случаях — самой удобной командой вывода:

■ ?/?? [<выр1>] [PICTURE <вырC1>] [FUNCTION <вырC2>]
[AT <вырN>] [,<выр2> ...]

Опции команды:

PICTURE/FUNCTION применяются для задания шаблонов и кодов управления выводом, которые перечислены при описании команды @...SAY...GET. Здесь введен дополнительный форматный код V<n>. Он организует вывод выражения в несколько строк с ограничением числа позиций по горизонтали <n> столбцами. Если значение <n> меньше длины слова, последнее "ломается" у правой границы, если нет — слова переносятся целиком. Этот код особенно удобен для вывода мемо-полей. При определении форматных функций в <вырC2> знак "@" не нужен.

AT — Номер столбца <вырN>, с которого должен начинаться вывод.

Команда ? выводит результаты выражений с новой строки, а ?? осуществляет выдачу данных на текущей строке в текущей колонке экрана.

Примеры:

```
.? 'ВСЕГО=' 3*2+0.5          && ВСЕГО= 6.50
?? 'рублей'                  && ВСЕГО= 6.50 рублей
.? 1/3 PICTURE('#.##') AT 4   && 0.33 (с 4-й позиции)
.? 'Оля' FUNCTION 'V1' AT 25, 'Покровская' FUNCTION 'V4' AT 34
                                0      Покр
                                л      овск
                                я      ая
```

Имя (Оля) выведено в одну колонку в 25-й позиции, фамилия (Покровская) — в четыре колонки с 34-й позиции.

Используя значение функции V1 для каждой из переменных, включенных в команды ?/??, легко организовать вертикальный вывод слов, причем следующая команда ? осуществит вывод ниже самого нижнего символа, выданного предыдущей командой. Это очень полезно при печати широких таблиц — заголовки колонок можно выводить в несколько строк.

Применяя шаблоны и форматные функции, можно выводить данные в разрядку:

```
.? 'Петров' PICTURE('@R x x x x x')          && П е т р о в
```

Следующая команда выведет в разрядку все фамилии из базы KADR.DBF (имя и отчество будут выданы обычным образом):

```
.USE kadr
.LIST OFF fam FUNCTION('@R'+REPLICATE(' x ',AT(' ',fam)-1)),;
SUBSTR(fam,AT(' ',fam))
```

Фамилией здесь считается все, что находится до первого пробела.

Командами ?/?? можно вывести не только видимые символы, но и звуковой сигнал (CHR-код — 7):

```
? 'Внимание !', CHR(7)
```

Очень часто нужно организовать предупреждающий сигнал, но так, чтобы не портилось имеющееся содержимое экрана. Команда ?CHR(7), хотя и не выводит никакого видимого символа, порождает на экране пустую строку, которая продвигает изображение на одну позицию вверх. В этом случае следует пользоваться командой ??CHR(7).

При выводе элементы данных, перечисленные в команде ?/??, будут отделены друг от друга одним пробелом. Если этого нужно избежать, можно

применить команду

■ SET SPACE OFF

По умолчанию SET SPACE ON.

Перенаправление вывода информации осуществляется командами

■ SET PRINTER ON/OFF

Установка команды SET PRINTER ON вызывает направление данных на принтер, SET PRINTER OFF — отмену (эта форма команды действует по умолчанию).

Данные можно направить и в другое место, используя команду

■ SET PRINTER TO [<файл> [ADDITIVE] / <порт>]

В частности, можно направить выдачу в указанный <файл>. Никакое расширение имени файла по умолчанию не подразумевается. При этом, если включено слово ADDITIVE и файл с таким именем уже существует, информация будет добавлена в конец файла. Если нет — файл будет переписан.

В качестве портов вывода можно указать любой из параллельных (LPT1, LPT2 или LPT3) или один из последовательных портов (COM1 или COM2). По умолчанию подразумевается <порт> — PRN, т.е. принтер.

Пример. Вывод данных в файл S.TXT.

SET PRINTER ON	&& Разрешение вывода
SET PRINTER TO s.txt	&& Назначение файла вывода S.TXT
? <генерация данных>	
SET PRINTER TO	&& Закрытие файла
SET PRINTER OFF	&& Запрещение вывода

Команда управления принтером ???

Коды управления принтером могут быть включены в команды ?/??, но в FoxPro имеется специальная команда управления принтером

■ ??? <вырС>

где <вырС> содержит эти коды.

Команда позволяет использовать коды принтера для его сброса, изменения типа и размера шрифта. Коды могут быть заданы в функции CHR() и/или в фигурных скобках {}.

Отличие в применении команд ? и ??? возникает при подключении внутреннего драйвера принтера FoxPro. В этом случае коды управления, включенные в команду ?/??, будут интерпретироваться этим драйвером по-своему, а коды из команды ??? прямо направляются на принтер.

Команда вывода TEXT

Структура

■ TEXT

<сообщения>

ENDTEXT

удобна для вывода значительных объемов текста, который выдается на экран/принтер (командой SET PRINTER ON) без всяких изменений <сообщений>, которые могут состоять из нескольких строк (например, сложные заголовки таблиц).

Команда вывода \

Команда

■ \ и \\ <строка текста>

выводит <строки текста>. Команда \ после вывода строки осуществляет переход в начало следующей строки, а команда \\ — нет. Команды похожи на команды ? и ??, но в некоторых отношениях они отличаются. <Строки> не нужно заключать в апострофы/кавычки. Среди текста выводимых данных могут содержаться также и выражения, которые окружаются разделителями << >>. При выводе эти выражения вычисляются.

Следующая команда определяет возможность вывода любых выражений в командах \, \\ и между командами TEXT и ENDTEXT:

■ SET TEXTMERGE [ON/OFF] [TO [<файл>] [ADDITIVE]] [WINDOW <окно>] [SHOW/NOSHOW]

Такие выражения должны быть взяты в разделители << >>. По умолчанию вывод запрещен (SET TEXTMERGE OFF).

Если вывод разрешен, он направляется на экран. Вывод возможен и в текстовый файл, если указана опция TO <файл>. Если такого файла нет, он будет создан, если есть, он будет перезаписан либо дополнен новыми данными при использовании опции ADDITIVE.

Если задана опция WINDOW <окно>, вывод можно направить в любое существующее <окно>. Вывод на экран/окно подавляется опцией NOSHOW. По умолчанию установлено SHOW.

В качестве выводимых данных допустимо использовать имена мемо-полей, взятые в разделители. В свою очередь, содержимое мемо-полей также может содержать выражения с разделителями.

Это очень удобно для предварительного формирования в мемо-полях шаблонов некоторых стандартных текстов, например писем, которые по мере необходимости выводятся с конкретным наполнением.

П р и м е р. Создадим базу стандартных писем PISMA.DBF, которая имеет три поля — символьное поле названия письма (NAZ), мемо-поле с заготовкой содержимого письма (SODER) и символьное поле, содержащее имя программы вывода писем (PROGR). Одна из записей такого файла пусть будет содержать средства создания стандартных форм поздравлений с днем рождения. В поле NAZ будет помещено название типа письма "Поздравление с днем рождения", в мемо-поле SODER — шаблон письма, приведенный ниже:

```
<<IIF(pol='M','Дорогой','Дорогая')>> <<fam>>
Поздравляем Вас с днем рождения,
```

```
<<DATE(>> Коллектив сотрудников
```

а в поле PROGR — имя процедуры POZDR.

Следующая программа позволит вам сначала выбрать необходимый тип письма в файле PISMA.DBF (BROWSE-окно используется как меню), а затем, если BROWSE-окно покинуто на этой записи, вывести письма для всех сотрудников (у кого день рождения сегодня) на экран и в текстовый файл POZDR.TXT с тем, чтобы в дальнейшем их напечатать. Имя нужной процедуры берется из поля PROGR и с помощью макроподстановки & включается в команду DO.

```
USE pisma IN A
BROWSE FIELD naz
a=(progr)
DO &a
```

```
&& Открытие базы PISMA.DBF
&& Выбор/создание текста письма
&& Определение имени процедуры
&& Вызов процедуры
```

```
PROCEDURE pozdr
```

```
&& Процедура вывода писем-поздравлений
```


SET TEXTMERGE ON TO pozdr	&& Открытие текстового файла POZDR.TXT
SELECT B	
USE kadr	&& Открытие базы KADR.DBF и отбор име-
SCAN FOR dtr=DATE()	&& нинников (дата рождения - сегодня)
\\ <<a.soder>>	&& Заполнение текстового файла
WAIT	&& Пауза для просмотра
ENDSCAN	
SET TEXTMERGE TO	&& Закрытие текстового файла
RETURN	

Далее текстовый файл может быть распечатан и поздравления разосланы. Одно такое возможное письмо-поздравление изображено ниже:

Дорогая РОМАНОВА М.С.
Поздравляем Вас с днем рождения,
09.10.92 Коллектив сотрудников

Функции IIF() и & и команда SCAN будут рассмотрены позже.

Вид левого и правого разделителей (<< и >>) может быть изменен на <вырC1> и <вырC2> командой.

■ SET TEXTMERGE DELIMITERS TO [<вырC1>,<вырC2>]]
в случае, если они сами используются как элементы текста.

Команда ввода INPUT

Команда выводит на экран сообщение-подсказку и ожидает ввода значения переменной.

■ INPUT <сообщение> TO <переменная>

Если переменная символьная, при вводе она должна быть взята в кавычки или апострофы.

Пример:

```
.INPUT 'Укажите месяц:' TO mes
Укажите месяц: 'Май'
.INPUT 'Зарплата=' TO z
Зарплата= 420
```

Команда ввода ACCEPT

Эта команда аналогична предыдущей, но предназначена для ввода только символьных переменных, и поэтому вводимая строка в апострофы не берется.

■ ACCEPT <сообщение> TO <символьная переменная>

Пример:

```
.ACCEPT 'Укажите город - ' TO gor
Укажите город Москва
```

Команды INPUT и ACCEPT являются рудиментарными средствами языка и сохраняются только для совместимости с ранними версиями пакета.

Команда ввода-вывода WAIT

Основная функция команды — приостановка программы, возможно, с вводом-выводом данных.

■ WAIT [<сообщение>] [TO <символьная переменная>]
[TIMEOUT <вырN>] [WINDOW [NOWAIT]] [CLEAR]

В зависимости от параметров команда выполняет действия:

WAIT — эта форма команды вызывает приостанов программы. На экране возникает фраза "Press any key to continue ..." ("Нажмите любую клавишу для продолжения ..."). После нажатия любой содержательной клавиши, клавиши Enter, клавиш со стрелками или перемещения маркера мыши на новое место работа программы возобновляется. Такая команда нужна, например, для того, чтобы приостановить и просмотреть изображение на экране.

WAIT <сообщение> — вместо указанного выше позволяет установить любое <сообщение>. Возобновление исполнения программы по-прежнему осуществляется любой клавишей. Иногда используется следующая разновидность команды — **WAIT "** (между апострофами нет пробела). Такая команда формально выводит на экран строку нулевой длины (ничего не выводит). Это бывает, например, удобно для приостановки выдачи данных на экран без разрушения его вида каким-либо техническим сообщением (даже и в окне системных сообщений).

WAIT <сообщение> TO <символьная переменная> — вызывает индикацию на экране <сообщения> и ожидание ввода только одного единственного символа в символьную переменную. После этого (без нажатия клавиши Enter) работа программы возобновляется. Данная форма может быть использована для принятия каких-либо решений. Например, с помощью команды

```
WAIT 'Будете продолжать (Д/Н)?' TO s WINDOW
```

можно организовать простейшее меню, пользуясь которым можно предусмотреть те или иные действия в зависимости от введенного в переменную S символа Д или Н.

Опции команды:

TIMEOUT <вырN> — команда ожидает ввода <вырN> секунд.

WINDOW [NOWAIT] — <сообщение> выводится не в текущий экран, а в системное окно сообщений в правом верхнем углу экрана. Если в команду включается опция NOWAIT, то клавишами, продолжающими исполнение программы, могут быть не только клавиши, перечисленные выше, но и любые управляющие клавиши, а также перемещение маркера мыши без нажатия ее кнопки. Кроме того, команда с этим словом не прерывает программу, хотя <сообщение> и остается на экране.

CLEAR — эта опция используется только со словом WAIT и предназначена для удаления с экрана сообщения, вызванного другой командой WAIT с опцией NOWAIT. Такая возможность очень полезна для того, чтобы показать пользователю, что, хотя на экране ничего не происходит, выполняется какой-то процесс, результатов которого нужно подождать. Это позволяет избежать нежелательных нажатий на клавиши. Например, типична такая последовательность команд:

```
WAIT 'Ждите, идут вычисления!' NOWAIT WINDOW
<команды>
WAIT CLEAR
```

Команда заполнения буфера клавиатуры

С помощью команды

■ **KEYBOARD <вырC>**

можно заполнить буфер клавиатуры строкой <вырC>. Таким образом

можно имитировать из программы ввод данных с клавиатуры, не делая его фактически. Эти данные затем будут считаны первой же командой, ожидающей ввода. В <вырС> можно помещать не только собственно данные, но и управляющие коды. Например, команда

```
KEYBOARD 'ПЕТРОВ'+CHR(13)
```

эквивалентна вводу с клавиатуры фамилии ПЕТРОВ и нажатию клавиши Enter. Нажатия клавиш можно указывать не только их кодами, но и клавишными именами, взятыми в апострофы/кавычки и фигурные скобки (см. команду ON KEY LABEL). Здесь CHR(13) можно заменить строкой '{ENTER}', что, конечно, гораздо удобнее.

Таким образом, возможно применение команды и для передачи содержательных данных в область редактирования (в командах CHANGE/EDIT, BROWSE, READ) из процедур.

Использование команды KEYBOARD позволяет, например, создать демонстрационный ролик для рекламы готовой работы.

Очищается буфер командой

■ CLEAR TYPEAHEAD

которая используется всегда, когда нужно иметь гарантированно пустой буфер.

Глава 13. РАБОТА С ПЕРЕМЕННЫМИ

Эффективное программирование возможно только при наличии аппарата временных переменных и массивов переменных.

В FoxPro разрешается иметь переменные тех же типов (кроме мемо), что и поля. Однако символьные переменные допускают гораздо большую длину — до 64 Кбайт, а числовые — представление и с плавающей точкой. Переменным и массивам переменных даются имена по тем же правилам, что и полям.

Кроме "обычных" переменных в FoxPro (подобно dBASEIV) введены так называемые системные переменные, которые являются резидентными и не могут быть уничтожены. Такие переменные имеют специальные имена, начинающиеся с символа "_". Системные переменные предназначены для запоминания некоторых установок среды FoxPro (в основном по управлению печатью). Из них мы рассмотрим лишь некоторые.

Команда присваивания

Следующая команда создает переменные и присваивает им значения.

■ **<переменная>=<выражение>**

или

■ **STORE <выражение> TO <имена переменных>**

Например, две команды идентичны: `a=c*(2+3)` и `STORE c*(2+3) TO a`. В обоих случаях переменной A присваивается значение `C*(2+3)`. Вторая форма команды предпочтительнее в случае, если нужно одно и то же значение присвоить сразу нескольким переменным.

Например: `STORE 0 TO f,d,c,r`

Тип переменной определяется типом последнего присваиваемого ей выражения. Символьные константы должны быть взяты в апострофы, кавычки или квадратные скобки. Например: `X='24'` и `Y=24`, где X — символьная, а Y — числовая переменные.

Команда может применяться и к массивам целиком или их элементам:

`DIMENSION d(10)`

`d='**'`

`STORE 5 TO d(1),d(2)`

В приведенном примере сначала весь массив D из десяти элементов (описанный командой DIMENSION) заполняется звездочками, а затем первым двум его элементам присваивается значение 5. (Работа с массивами будет рассмотрена позже.) При желании переменные и массивы могут быть сохранены в файлах типа MEM или мемо-полях и загружены из них в память, а также уничтожены или предъявлены на экране/принтере.

Сохранение переменных

Следующая команда сохраняет в <файле> или <мемо-поле> с заданным именем все или часть из имеющихся к текущему моменту переменных:

■ **SAVE TO <файл>/TO MEMO <мемо-поле>**

[ALL LIKE/EXCEPT <маска>]

Если задано ALL LIKE, то сохраняются только переменные, соответствующие маске, если ALL EXCEPT — все, за исключением соответствующих маске. По умолчанию имени файла придается расширение MEM. Маска может содержать знаки, принятые в MS DOS: "?" и "*".

Примеры сохранения переменных в файле FFF.MEM:

SAVE TO fff	- сохраняются все переменные;
SAVE TO fff ALL LIKE a*	- сохраняются переменные, начинающиеся с буквы A;
SAVE TO fff EXCEPT ??d*	- сохраняются все переменные, кроме тех, которые в качестве третьей буквы имени имеют D;
USE kadr	
SAVE TO MEMO per	- сохраняются все переменные в мемо-поле PER базы KADR.DBF.

Если файл, в котором вы хотите сохранить переменные, уже существует, FoxPro сообщит об этом строкой

<ФАЙЛ> already exists, overwrite it?

т.е. "<ФАЙЛ> уже существует, перезаписать его?". Выбором пункта меню Yes файл будет обновлен. В программе такие предупреждающие сообщения обычно подавляются (по умолчанию ON) командой

■ SET SAFETY OFF

Команда действует при перезаписи не только MEM-файлов, но и всех других типов файлов, а также при очистке баз данных и некоторых других необратимых действиях.

Загрузка переменных в память

При необходимости работать с ранее сохраненными переменными используется команда

■ RESTORE FROM <файл> /FROM MEMO <мемо-поле> [ADDITIVE]

Команда загружает в память все переменные из указанного файла типа MEM или <мемо-поля>. Все имеющиеся в памяти к этому времени переменные уничтожаются. Чтобы этого избежать, можно включить фразу ADDITIVE.

П р и м е р: RESTORE FROM fff ADDITIVE

Удаление временных переменных

Активное использование временных переменных позволяет строить эффективные и быстрые системы обработки данных. Однако возможности сохранения в памяти компьютера значительного числа таких переменных ограничены ее емкостью. Ввиду этого память следует при необходимости периодически очищать от ненужных в данный момент или сохраненных в MEM-файлах переменных командой RELEASE:

■ RELEASE <переменные>/RELEASE ALL [LIKE/EXCEPT <маска>]

Команда удаляет только указанные <переменные>, или все (ALL), или соответствующие (LIKE), или не соответствующие (EXCEPT) <маске>. Уничтожение всех переменных может быть выполнено также командой

■ CLEAR MEMORY

Просмотр переменных

В любой момент при отладке программ может понадобиться просмотр переменных памяти. Использование команды "?" слишком трудоемко при просмотре большого количества переменных. Следующая команда предъявляет сразу все нужные переменные памяти:

■ DISPLAY MEMORY [LIKE <маска>] [TO PRINTER/FILE <файл>]

Команда показывает с паузами после выдачи каждых 20 строк все активные переменные и массивы, их статус (PUBLIC или PRIVATE), тип и значения. Кроме того, она показывает размер памяти, занятой под переменные, окна и меню. Если нужно просмотреть только все пользовательские переменные, введите команду DISPLAY MEMORY LIKE *. Аналогичные функции, но без остановок выполняет команда LIST MEMORY.

Все вышеперечисленные в этом разделе команды, кроме команд просмотра, не распространяются на системные переменные FoxPro.

Глава 14. КОМАНДЫ УПРАВЛЕНИЯ

Команды управления являются важнейшим средством построения программ. Эти команды не могут быть опробованы и использованы в интерактивном режиме, а только в программах.

Команда IF. В зависимости от условия команда выполняет те или иные <команды>, находящиеся внутри конструкции IF...ENDIF.

```
■ IF <условие>
    <команды>
[ELSE
    <команды>]
```

ENDIF

Если условие истинно, выполняются все <команды>, следующие от IF до ELSE, если ложно, то <команды> от ELSE до ENDIF. Если необязательная фраза ELSE отсутствует и условие ложно, все внутренние <команды> пропускаются и выполняется команда, следующая за ENDIF. Допустимо вложение друг в друга конструкций типа IF ... ENDIF и других структурных команд.

П р и м е р. Вывести большее из двух чисел А и В, а если числа равны, вывести сообщение "ЧИСЛА РАВНЫ".

```
IF a=b
    ? 'ЧИСЛА РАВНЫ'
ELSE
    IF a>b
        ? a
    ELSE
        ? b
    ENDIF
ENDIF
```

Замечание. В одной строке программы разрешается записывать только одну команду, и ее положение в строке произвольно. Для наглядности лучше записывать их не в столбец, а уступами, подчеркивающими вложенность команд.

Команда DO CASE. Конструкция DO CASE ... ENDCASE решает задачи, аналогичные команде IF, но в ней может быть указано сразу несколько условий, которые последовательно проверяются во всех фразах CASE.

```
■ DO CASE
    CASE <условие 1>
        <команды>
    CASE <условие 2>
        <команды>
    .
    .
    .
[OTHERWISE
    <команды>]
```

ENDCASE

Если встретилось истинное <условие>, выполняются нижеследующие <команды> до следующей фразы CASE, или OTHERWISE, или ENDCASE, и конструкция завершается. Если ни одно из CASE-условий не истинно, выполняются <команды>, стоящие за фразой OTHERWISE до ENDCASE, если фраза OTHERWISE отсутствует, не выполняется ни одна команда.

Пример для условий предыдущей задачи показан ниже.

```
DO CASE
    CASE a=b
        ? 'ЧИСЛА РАВНЫ'
    CASE a>b
        ? 'a=' , a
    CASE a<b
        ? 'b=' , b
    ENDCASE
```

Команда очень удобна для обработки выбора из меню в программах. Разрешается вложение команд DO CASE, IF.

В случае, если найдено истинное CASE-условие, остальные условия не проверяются и выполняется команда, стоящая за ENDCASE. Кроме перечисленных команд IF и DO CASE, в СУБД FoxPro имеется очень полезная функция анализа условия — IIF() (см. гл.16).

Глава 15. ОРГАНИЗАЦИЯ ЦИКЛОВ

В FoxPro имеются развитые средства организации программных циклов, которые в зависимости от постановки задачи могут быть классифицированы на циклы с условием (итерационные циклы) и циклы с параметром (арифметические циклы).

Цикл с условием. Реализация так называемых итерационных циклов, т.е. циклов с заранее известным условием их окончания, выполняется следующей конструкцией:

```
■ DO WHILE <условие>
    <команды>
ENDDO
```

Команды, заключенные между DO WHILE и ENDDO, будут выполняться до тех пор, пока <условие> истинно. Если оно ложно или становится ложным, осуществляется переход к команде, следующей за ENDDO.

П р и м е р. Ввести с клавиатуры и суммировать числа X в переменную S до тех пор, пока сумма не превысит 1000.

```
s=0
DO WHILE s<=1000
    INPUT 'ВВЕДИТЕ X' TO x
    s=s+x
ENDDO
```

В языке FoxPro отсутствует понятие "метка", т.е. нет естественной возможности перейти в другое место программы. Это создает некоторые (кажущиеся) сложности, которые можно преодолеть. Так, если требуется выйти за пределы цикла, необходимо использовать команду

■ EXIT

которая передаст управление команде, следующей за ENDDO.

Следующая команда осуществляет передачу управления в цикле, но в противоположную сторону — в его начало, на саму команду цикла:

■ LOOP

Это нужно, чтобы при необходимости избежать выполнения некоторых команд, предшествующих фразе END, и сразу перейти к следующему циклу.

Рассмотренные команды действуют не только в структуре DO WHILE, но и во всех других командах организации циклов (FOR и SCAN).

П р и м е р. Допустим, что в условиях предыдущего примера требуется суммировать только положительные числа X, а в случае, если встретится X=0, вообще прекратить суммирование.

```
s=0
DO WHILE s<1000
    INPUT 'ВВЕДИТЕ X' TO x
    IF x<0
        LOOP
    ENDIF
    IF x=0
        EXIT
    ENDIF
    s=s+x
ENDDO
```

Отсутствие меток вынуждает программиста обращаться к командам EXIT и LOOP, которые, однако, имеют смысл только в циклах. Поэтому иногда приходится использовать команды цикла там, где никакого цикла нет, применяя их как операторные скобки. В качестве условия ставится какое-либо всегда истинное условие, обычно это просто логическая "Истина" (.T.):

```
DO WHILE .t. <команды> ENDDO.
```


Цикл с параметром. Арифметический цикл предполагает наличие переменной, ограничивающей число циклов.

■ FOR <переменная>=<вырN1> TO <вырN2> [STEP <вырN3>]
 <команды>
 ENDFOR

Здесь <переменная> используется в качестве управляющего параметра цикла, для которого <вырN1> является начальным значением, <вырN2> — конечным, а <вырN3> — шагом изменения <переменной>. Если последний параметр отсутствует, шаг равен 1.

Таким образом, цикл будет выполняться столько раз, сколько нужно, чтобы <переменная> от значения, равного <вырN1>, достигла <вырN2> с шагом <вырN3>. <Переменная> в каждом цикле сравнивается с <вырN2>. Если она меньше или равна <вырN2>, продолжается выполнение цикла, если больше, выполнение цикла заканчивается и программа продолжается с команды, следующей за ENDFOR. Если <вырN3> отрицательно, переменная в каждом цикле будет уменьшаться. Тогда <вырN1> должно быть больше <вырN2>.

Хотя все параметры команды FOR допускаются менять внутри цикла, это никак не влияет на число циклов или значение <переменной>. Исключение составляет сама <переменная>, изменение которой влечет изменение числа циклов. В цикл FOR могут включаться команды EXIT и LOOP.

Наиболее частое применение команды FOR — организация циклов с заранее известным их числом. Тогда <переменная> имеет смысл счетчика циклов.

П р и м е р ы. Значения переменной I показаны после знаков &&.

FOR i=2 TO 5	FOR i=12 TO 3 STEP -2
? i && 2,3,4,5	? i && 12,10,8,6,4
ENDFOR	ENDFOR
? i && 6	? i && 2

Допускается также использование вместо команды ENDFOR команды NEXT (как в Бейсике).

Цикл сканирования базы данных. Следующая команда применяется для перемещения в базе данных и выполнения <команд> для каждой встреченной записи, которая отвечает условиям.

■ SCAN [<границы>] [FOR <условие>] [WHILE <условие>]
 <команды>
 ENDSCAN

При отсутствии границ и условий сканируется вся база данных. Команда SCAN является исключительно удобным средством перемещения в базе.

Здесь уместно обсудить вопрос о технике перемещения в базе данных "по условию". Ниже приводятся фрагменты программ поиска в базе KADR.DBF всех записей с фамилиями, начинающимися с буквы "П", как с использованием цикла DO WHILE, так и SCAN-цикла. Рассматривается как последовательный поиск, так и ускоренный с применением индексного файла KADRFAM.IDX, созданного по полю FAM.

- | | |
|---|---|
| 1. USE kadr
LOCATE FOR fam='П'
DO WHILE !EOF()
<обработка записи>
CONTINUE
ENDDO | 2. USE kadr [INDEX kadrfam]
SCAN FOR fam='П'
<обработка записи>
ENDSCAN |
| 3. USE kadr INDEX kadrfam
SEEK 'П'
DO WHILE fam='П'
<обработка записи> | 4. USE kadr INDEX kadrfam
SEEK 'П'
SCAN WHILE fam='П'
<обработка записи> |

SKIP
ENDDO

ENDSCAN

В примере 2 подключение индекса (опция INDEX kadrfram) обеспечивает сканирование базы с оптимизацией по технологии Rushmore. В этом случае пример идентичен примеру 4. Однако для оптимизирующей технологии лучше, если главный индекс не назначен (ORDER 0).

Очевидно, что использование SCAN-цикла существенно удобнее и быстрее. Однако это не исключает ситуации, когда имеет смысл применение для указанных целей и цикла DO WHILE, в особенности если нам нужно самим управлять перемещением указателя записей.

Следует заметить, что выполнение каждой команды ENDSCAN перемещает указатель записей. Это значит, что завершение SCAN-цикла с условием переносит нас за пределы действия этого условия. Пусть нам нужно найти и вывести суммарную зарплату S для каждого подразделения базы KADR.DBF (KADRPODR.IDX — индексный файл по полю PODR). Использование двух вложенных SCAN-циклов (ниже слева) повлечет ошибку — пропуск одной записи для каждого нового подразделения. Ниже справа приведено правильное решение.

```
USE kadr INDEX kadrpodr
SCAN
  p=podr
  s=0
  SCAN WHILE podr=p
    s=s+szar
  ENDSCAN
  ?p ,s
ENDSCAN
```

```
USE kadr INDEX kadrpodr
DO WHILE !EOF()
  p=podr
  s=0
  SCAN WHILE podr=p
    s=s+szar
  ENDSCAN
  ?p ,s
ENDDO
```

В заключение рассмотрим одну специфическую команду — команду выполнения внешних по отношению к FoxPro программ

■ !/RUN <программа>

При вводе этой команды в свободную память компьютера загружается и выполняется желаемая <программа>, например команда DOS. Если места для внешней программы недостаточно, может быть автоматически осуществлена частичная выгрузка FoxPro на диск до завершения программы. Размер доступной памяти под RUN-программы можно запросить с помощью функции MEMORY() или SYS(12).

Пример: .! DEL *.txt>NUL

В текущей директории удаляются все файлы с расширением TXT. Такая команда в некоторых случаях может быть удобнее, чем собственная команда FoxPro, для уничтожения (или копирования) сразу многих файлов, поскольку позволяет использовать символы маски "*" и "?". Чтобы избежать вывода на экран ненужных нам возможных системных сообщений, они направляются на "нулевое" устройство DOS — NUL.

Эту команду можно использовать, например, для установки текущей системной даты из прикладной программы

```
.d='03.08.93'
.!DATE &d
.?DATE()
03.08.93
```

При этом только следует иметь в виду формат даты, принятый в DOS, поскольку он может не совпадать с форматом, установленным в FoxPro.

Глава 16. ФУНКЦИИ СУБД

Функции в FoxPro используются для анализа или преобразования данных. Синтаксическая особенность функций — обязательное наличие скобок (кроме функции &).

Все функции могут быть использованы в программах, а большинство из них — и в интерактивном режиме. Изучение функций следует начинать, просто вводя их в командном окне со знаком вопроса. Все приведенные примеры реализованы таким образом. Результаты выполнения примеров показаны справа после знаков комментария "&&".

Функции разбиты (иногда довольно условно) на следующие группы:

- математические функции;
- строковые функции;
- функции работы с датами;
- функции преобразования типов данных;
- функции проверки файлов и дисков;
- функции позиционирования выдачи данных;
- функции работы с мышью;
- клавишные функции;
- технические функции;
- функции времени;
- функция анализа условий;
- функции анализа типа и наличия данных;
- финансовые функции;
- функции подстановки.

Здесь разобрано большинство функций FoxPro. Однако некоторые только перечислены и подробно рассматриваются в соответствующих разделах. Описание оконных функций, функций работы с массивами, функции меню и часть функций, связанных с индексированием, отнесено в соответствующие разделы.

Математические функции

Арифметические функции

- **ABS(<вырN>)** — вычисляет абсолютное значение <вырN>.

Пример:

```
.?ABS(-24.8)
```

&& 24.8

- **BETWEEN(<выр>,<выр1>,<выр2>)** — возвращает значение "Истина" (.T.) если <выр> больше или равно <выр1> и меньше или равно <выр2>, иначе — "Ложь" (.F.). Тип всех трех выражений должен быть одинаковым (строка, число, дата).

Пример:

```
.? BETWEEN(4,1,6)
```

&& .F.

```
.? BETWEEN({04.06.65},{23.10.70},{15.12.91})
```

&& .T.

```
.USE kadr
```

```
.LIST FOR BETWEEN(YEAR(dtr), 1943,1960) fam,dtr
```

Последняя команда выводит все записи из базы KADR.DBF, где дата рождения находится между 1943 и 1960 годами.

- **CEILING(<вырN>)** — возвращает ближайшее целое число большее или равное <вырN>. Аргумент может иметь любой знак.

Пример:

```
.? CEILING(6.3), CEILING(-8.4)
```

&& 7 и -8

- **FLOOR(<вырN>)** — ближайшее целое меньшее или равное <вырN>.

Пример:

```
.? FLOOR(6.3), FLOOR(-8.4)
```

&& 6 и -9

- **INT(<вырN>)** — целая часть <вырN>.

Пример:

```
.? INT(-18.7)
```

&& -18

- **MAX(<выр>,<выр1>[,<выр2> ...])** — возвращает максимальное значение из

списка аргументов, которые должны быть все одного типа (символьные, числовые или дата):

Пример:

? MAX(3, 1, -8) && 3
 ? MAX({04.06.65}, {23.10.70}, {15.12.91}) && 15.12.91

- MIN(<выр>, <выр1>[, <выр2> ...]) — возвращает минимальное значение из списка аргументов, которые должны быть все одного и того же типа.

Пример:

? MIN(3, 1, -8) && -8
 ? MIN({04.06.65}, {23.10.70}, {15.12.91}) && 04.06.65

- MOD(<вырN1>, <вырN2>) — целочисленный остаток от деления <вырN1> на <вырN2>.

Пример:

? MOD(3, 2), MOD(5/7) && 1 и 5

- ROUND(<вырN1>, <вырN2>) — округление <вырN1> до заданного в <вырN2> количества знаков после запятой.

Пример:

? ROUND(-342.268, 1) && -324.3

- RAND([<вырN>]) — возвращает псевдослучайное число в диапазоне 0 — 1. <ВырN> позволяет определить начальное значение аргумента функции. Использование одних и тех же значений <вырN> дает один и тот же ряд чисел. По умолчанию исходное значение аргумента 100001, что соответствует функции RAND(100001). Если <вырN> является отрицательным, тогда значение функции вычисляется с помощью таймера компьютера, что обеспечивает наибольшую случайность выдаваемых значений. Таким образом, для получения максимальной случайности выдаваемых значений вначале следует использовать функцию с отрицательным аргументом, а затем — без аргументов. Хотя сама функция непосредственно вырабатывает числа, распределенные равномерно между 0 и 1, на ее основе можно построить генераторы с любыми диапазонами чисел и любыми законами распределения. Например, выражения

$$(b-a)*RAND()+a \text{ и } INT((j-i+1)*RAND()+i)$$

реализуют равномерный закон с вещественными числами, расположенными в диапазоне A — B, и с целыми числами в диапазоне I — J. Генераторы других законов приведены в документации на FoxPro.

- SIGN(<вырN>) — возвращает значения: 1, если число положительное, -1, если отрицательное, 0, если нуль.

Пример:

? SIGN(5), SIGN(-7), SIGN(0) && 1, -1 и 0

Степенные функции

- EXP(<вырN>) — экспонента <вырN> — основание натурального логарифма "е" в степени <вырN>.
- LOG(<вырN>) — натуральный логарифм <вырN>. Аргумент должен быть больше нуля.
- LOG10(<вырN>) — десятичный логарифм <вырN>. Аргумент должен быть

больше нуля.

- **SQRT(<вырN>)** — квадратный корень <вырN>. Аргумент должен быть положительным.

Тригонометрические функции

- **SIN(<вырN>)** — синус(<вырN>). <ВырN> задается в радианах. Результат, возвращаемый функцией, находится в диапазоне -1 — +1.
- **COS(<вырN>)** — косинус(<вырN>). Аргумент задается в радианах.
- **TAN(<вырN>)** — тангенс(<вырN>). Аргумент задается в радианах..
- **ASIN(<вырN>)** — арксинус(<вырN>). Результат в радианах и в интервале от $-\pi/2$ до $\pi/2$ (от -1,57079 до 1,57079). Значение аргумента может изменяться от +1 до -1(<вырN>). Результат в радианах в интервале от 0 до π (3,14159). Аргумент может изменяться от -1 до +1.
- **ATAN(<вырN>)** — арктангенс(<вырN>). Результат в радианах в интервале от $-\pi/2$ до $\pi/2$. Значение <вырN> может быть любым.
- **ATN2(<вырN1>,<вырN2>)** — арктангенс отношения <вырN1>/<вырN2>. Здесь аргументы — это координаты точки Y и X на плоскости. Функция допускает 0 в качестве второго аргумента. Результат лежит в интервале между $-\pi/2$ и $\pi/2$.

П р и м е р:

.? ATN(5,0) && 1.57

- **PI()** — число π (приблизительно 3.141592).
- **DTOR(<вырN>)** — преобразует угол, заданный в градусах, в радианы.

П р и м е р:

.? DTOR(90), DTOR(40.2) && 1.57 и. 0.7

- **RTOD(<вырN>)** — возвращает угол в градусах по его радианному значению <вырN>. И аргумент, и результат функций DTOR() и RTOD() — в десятичном представлении.

П р и м е р:

.? RTOD(1.57) && 89.95

Здесь мы как-будто видим ошибку: 90 градусов соответствуют 1.57 радианам, а 1.57 радиан — только 89.95 градусам. На самом деле это является следствием не ошибки, а ограниченной точности вывода результата (по умолчанию — два дробных разряда) функции DTOR(). Если бы мы вывели не два, а больше разрядов, совпадение было бы гораздо более полным.

Число десятичных разрядов, выдаваемых после запятой, может быть установлено командами вида SET. По умолчанию число выводимых разрядов определяется разрядностью операндов. Если введена команда

■ SET FIXED ON

(по умолчанию OFF), то разрядность будет уже определяться другой командой:

■ SET DECIMALS TO <вырN>

устанавливающей количество отображаемых десятичных разрядов равным <вырN>, и при этом производится округление. По умолчанию <вырN>=2.

П р и м е р ы. Пусть выводимое число равно 67.34567. Тогда применение указанных команд даст следующие результаты:

.SET FIXED ON
.SET DECIMALS TO 3

&& результат: 67.35
&& результат: 67.346

Команды SET DECIMALS и SET FIXED управляют только предъявлением информации, но не влияют на ее фактическое значение.

Строковые функции

Функции анализа

- **<вырC1>\$<вырC2>** — возвращает .Т., если <вырC1> содержится в <вырC2>, и .Ф. в противном случае.

Пример:

.? 'база'\$'база данных' && .Т.

- **AT/ATC(<вырC1>,<вырC2>[,<вырN>])** — поиск слева направо в строке <вырC2> номера позиции, с которой начинается подстрока <вырC1>. Если подстрока не найдена, выдается 0. Параметр <вырN> указывает, какой экземпляр <вырC1> разыскивается в <вырC2>. По умолчанию <вырN>=1.

Пример:

.? AT('база','база данных') && 1
.? AT('а','база данных') && 2
.? AT('а','база данных',3) && 7
.? AT('хх','xxxxxxxx',3) && 3

Полностью аналогична функции AT() функция ATC(), которая при поиске не различает строчные и прописные буквы.

- **RAT(<вырC1>,<вырC2>[,<вырN>])** — поиск справа налево в <вырC2> номера позиции, с которой начинается <вырC1>. Если подстрока не найдена, выдается 0. <ВырN> — номер разыскиваемого экземпляра <вырC1>. Функция RAT() обратна функции AT().

Пример:

.? RAT('база','база данных') && 1
.? RAT('а','база данных') && 7
.? RAT('а','база данных',3) && 2
.? RAT('хх','xxxxxxxx',3) && 5

Хотя поиск ведется справа налево, найденный номер позиции отсчитывается нормально — слева направо.

- **INLIST(<выр>,<выр1>,<выр2> ...)** — возвращает .Т., если <выр> содержится в списке <выр1>,<выр2>,.... Все выражения должны быть выражениями одного типа (строки, числа, даты).

Пример:

.? INLIST('июль','июнь','июль','август') && .Т.

- **ISDIGIT(<вырC>)** — возвращает .Т., если первый символ <вырC> — цифра.

Пример:

.? ISDIGIT('1991 год') && .Т.

- **LEN(<вырC>)** — число символов в <вырC>. Длина нулевой строки (') — 0.

Пример:

.? LEN('база') && 4

- **LIKE(<вырC1>,<вырC2>)** — возвращает логическое значение .Т. или .Ф., которое указывает, имеется или нет <вырC1> в <вырC2>. Для <вырC1>

разрешается использовать символы маски * и ?. Функция очень удобна для поиска в базе по неполному ключу.

Пример:

```
.? LIKE('?аза данных', 'база данных')      && .Т.
.? LIKE('???????нных', 'база данных')        && .Т.
.? LIKE('*нных', 'база данных')               && .Т.
```

Пример. Пусть в некоторой базе данных в поле NOM хранятся номерные знаки автомобилей в формате БББ-ЦЦ-ЦЦ (Б — буквы, Ц — цифры). Из номера разыскиваемого автомобиля известны только его первые две из трех букв (МО) и две последние цифры (43), т.е. МО?-??-43. Нужно вывести все номера, удовлетворяющие этой информации:

. LIST nom FOR LIKE('МО?-??-43',nom)

Если искомая строка может содержаться в некотором тексте, ключ поиска необходимо ограничить знаками "*", т.е. '*МО?-??-43*'. Если положение цифр 43 в номерном знаке неизвестно, аргумент функции должен быть задан как 'МО?-*43*'.
 ■ OCCURS(<вырС1>,<вырС2>) — возвращает число вхождений <вырС1> в <вырС2>.

Пример:

```
.? OCCURS('а', 'база данных')                && 3
```

Функции выделения

- LEFT(<вырС>,<вырN>) — выделение из строки <вырС> указанного в <вырN> числа символов слева. Если <вырN> длиннее строки, возвращается вся строка.

Пример:

```
.? LEFT(4, 'база данных')                      && база
```

- RIGHT(<вырС>,<вырN>) — выделение из <вырС> справа <вырN> символов.

Пример:

```
.? RIGHT(6, 'база данных')                     && данных
```

- SUBSTR(<вырС>, <начальная позиция>, [<число символов>]) — выделение из <вырС> подстроки, начиная с <начальной позиции> длиной в <число символов>. Если последний параметр отсутствует, выделяемая подстрока заканчивается концом строки <вырС>.

Пример:

```
.? SUBSTR('база исходных данных', 6, 8)         && исходных
```

Функции преобразования

- CHRTRAN(<вырС1>,<вырС2>,<вырС3>) — конвертирует символы из <вырС1>, используя символы строк <вырС2> и <вырС3> как таблицы перевода. Все появления в <вырС1> первого знака из <вырС2> заменяются первым знаком из <вырС3>, второго знака из <вырС2> — вторым знаком из <вырС3> и т.д. Если <вырС3> имеет меньше знаков, чем <вырС2>, дополнительные знаки в <вырС2> переводятся в знаки нулевой длины. Если <вырС3> имеет больше знаков, чем <вырС2>, дополнительные знаки игнорируются.

Пример:

```
.? CHRTRAN("abcdef", "ace", "xyz")             && xbydzf
.? CHRTRAN("abcd", "ac")                        && bd
```


Функция может быть, например, использована для конвертирования строчных русских букв в прописные. Для этого введем две строковые переменные: одну для хранения строчных букв алфавита (S), другую — для прописных (P).

```
.S='абвгдежзийклмнопрстуфхцчшщъьэюя'
.P='АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЬЭЮЯ'
.? CHRTRAN('База Данных',S,P)                && БАЗА ДАННЫХ
```

- LTRIM(<вырС>) — удаление ведущих пробелов в <вырС>.
- TRIM/RTRIM(<вырС>) — удаление завершающих пробелов в <вырС>.
- ALLTRIM(<вырС>) — удаление всех: и ведущих и завершающих пробелов в строке <вырС>.
- REPLICATE(<вырС>,<вырN>) — <вырС> повторяется <вырN> раз.

Пример:

```
.? REPLICATE('+-',5)                && +-+--+---+
```

- SPACE(<вырN>) — формирование строки пробелов длиной <вырN>.
- STUFF(<вырC1>,<начальная позиция>,<число символов>,<вырC2>) — изменение любой части <вырC1> на <вырC2> с места, указанного <начальной позицией> длиной в заданное <число символов>. Если <число символов> равно 0, то <вырC2> вставляется в <вырC1>, не изменяя ни одного символа в исходной строке. Если <вырC2> является пустой строкой длиной 0, т.е. <вырC2>='', из исходной строки удаляется указанное <число символов> со сжатием <вырC1>.

Пример:

```
.X='база исходных данных'
.? STUFF(X,6,9,'конечных ')          && база конечных данных
.? STUFF(X,6,9,'')                  && база данных
```

- STRTRAN(<вырC1>,<вырC2>[,<вырC3>][,<вырN1>][,<вырN2>]) — возвращает <вырC1>, в котором встретившиеся в нем <вырC2>, замещены на <вырC3>. Замещение начинается с <вырN1>-го вхождения и продолжается до <вырN2>-го вхождения. Если <вырC3> опущено, <вырC2> будет превращено в пустую строку. Если не указано <вырN1>, оно считается 1, если не указано <вырN2>, все встретившиеся <вырC2> будут замещены.

Пример:

```
.? STRTRAN('база данных','a','A')    && БАЗА ДАННЫХ
.? STRTRAN('база данных','n','')      && база даых
.? STRTRAN('база данных','a','*',2)    && баз* д*нных
```

- TRANSFORM(<выр>,<шаблон>) — функция позволяет задать <шаблон> выдачи данных без команды @ ... SEY.

Пример:

```
.? TRANSFORM(234.183,'####.#')        && 234.1
```

- PADC/PADL/PADR(<вырC1>,<вырN>[,<вырC2>]) — возвращают строку, где <вырC1> вставлено соответственно в центр/слева/справа строки, составленной из <вырC2>, повторенного <вырN> раз. Если <вырC2> опущено — берется символ пробела. Если длина <вырC1> больше <вырN>, то <вырC1> усекается справа до заданной длины. Итоговая длина строки остается равной <вырN>.

Пример:

```
.? PADC('Таблица',20,'*')            && *****Таблица*****
```


. ? PADL('Таблица', 20, '*')

&& Таблица*****

Функции обработки мемо-полей

Из рассмотренных выше функций с мемо-полями могут использоваться LEN(), AT(), ATC(), SUBSTR(). Кроме того, следующие функции работают только с мемо-полями:

- ATLINE/ATCLINE(<вырС1>, <мемо-поле>) — ищет первое вхождение <вырС1> в мемо-поле и возвращает номер строки, где оно было найдено. Если <вырС1> не найдено, возвращается 0. Далее найденная строка может быть, например, взята на обработку функцией MLINE(). Совершенно аналогична функции ATLINE() функция ATCLINE(), которая не делает различий между прописными и строчными буквами.
- RATLINE(<вырС1>, <мемо-поле>) — ищет <вырС1>, определяя его последнее появление в <мемо-поле>, и возвращает номер строки мемо-поля, где найдено искомое выражение. Если оно не найдено, будет возвращен нуль.
- MLINE(<мемо-поле>, <вырN>) — возвращает строку номер <вырN> из мемо-поля. Если <вырN> больше числа строк, возвращается пустая строка.
- MEMLINES(<мемо-поле>) — возвращает число строк в мемо-поле. Длина строки определяется командой SET MEMOWIDTH (по умолчанию 50 символов). Функция полезна при печати отчетов с мемо-полями.

При разбиении функциями MLINE() и MEMLINES() мемо-поля на строки учитываются и уже имеющиеся разбиения строк, сделанные с помощью клавиши Enter. Считается, что этот символ (CHR(13)) завершает текущую строку, даже если она короче установленной командой SET MEMOWIDTH.

Функции конвертирования и распознавания букв

В FoxPro имеется группа символьных функций анализа и преобразования строчных/прописных букв строки.

- ISALPHA(<вырС>) — возвращает .Т., если <вырС> начинается с буквы.
- ISLOWER(<вырС>) — .Т., если <вырС> начинается со строчной буквы.
- ISUPPER(<вырС>) — .Т., если <вырС> начинается с прописной буквы.
- LOWER(<вырС>) — преобразует все прописные буквы <вырС> в строчные.
- PROPER(<вырС>) — преобразует в <вырС> первую букву каждого слова в прописную, а остальные в строчные. Удобна, например, для ввода фамилии, имени, отчества.
- UPPER(<вырС>) — преобразует все строчные буквы <вырС> в прописные.

В оригинальном пакете FoxPro эти функции действуют лишь с латинскими буквами, но при наличии русификатора работают и с кириллицей. Это же относится к функциям ATC() и ATCLINE().

Функции работы с датами

- CDOW(<вырD>) — возвращает имя дня недели по-английски.

Пример:

```
. SET DATE GERMAN
. ? CDOW({05.10.91})
```

&& Saturday

- **CMONTH(<вырD>)** — предъявляет название месяца по-английски.

Пример:

```
.? CMONTH({05.10.91})                                && October
```

- **CTOD(<вырC>)** — преобразует дату из символьной формы в формат даты (типD). Если не изменено командой SET DATE, строка <вырC> должна иметь американский формат — ММ/ДД/ГГ. Допускается, чтобы <вырC> лежало в диапазоне '1/1/1000' — '12/31/9999'. Функция используется для создания в программе переменных типа дата.

Пример:

```
.? CTOD('05.10.91')                                    && 05.10.91 типа дата
```

- **{<вырC>}** — функция преобразования строковой константы (без кавычек) в дату. Аналогична CTOD(), но более удобна. Однако в качестве аргумента не допускаются переменные и функции.

Пример:

```
.? {05.10.91}                                           && 05.10.91 типа дата
```

- **DATE()** — предъявляет системную дату, т.е. дату, которая была введена при загрузке в компьютер операционной системы. Формат предъявления зависит от команд SET CENTURY и SET DATE.

- **DAY(<вырD>)** — число месяца по выражению типа дата <вырD>.

Пример:

```
.? DAY({05.10.91})                                     && 5
```

- **DMY(<вырD>)** — возвращает по-английски в виде строки выражение даты из <вырD> в последовательности день, месяц, год. Если SET CENTURY OFF дата отображается в формате ДД МЕСЯЦ ГГ, если ON — ДД МЕСЯЦ ГГГГ.

Пример:

```
.? DMY({05.10.91})                                     && 05 October 91
```

- **DOW(<вырD>)** — определяет номер дня недели по <вырD>. Первый день недели — воскресенье, седьмой — суббота.

Пример:

```
.? DOW({05.10.91})                                     && 7
```

- **DTOC(<вырD>[,1])** — преобразует выражение типа дата <вырD> в символьное выражение. Необязательный параметр "1" означает, что формируемая строка будет не символьным отображением даты в каком-либо формате даты, а сплошной строчкой из четырех цифр года, двух цифр месяца и дня — ГГГГММДД. Такое изображение даты удобно при индексировании с составным ключом, в котором есть дата.

Пример:

```
.? DTOC({05.10.91})                                     && 05.10.91
.? DTOC({05.10.91},1)                                   && 19911005
```

- **DTOS(<вырD>)** — возвращает 8-разрядную символьную строку в формате ГГГГММДД для <вырD>. Аналогична DTOC() с параметром 1.

- **GOMONTH(<вырD>,<вырN>)** — возвращает дату, которая отстоит от <вырD> на <вырN> месяцев. Если <вырN> положительно, возвращаемая дата будет на <вырN> месяцев позже, если отрицательно — раньше. Если такой даты не существует, возвращается дата, соответствующая последнему дню вычисленного месяца.

Пример:

```
? GOMONTH({31.10.91},12)      && 31.10.92
? GOMONTH({31.10.91},-6)      && 30.04.91
```

В последнем случае функция возвращает "30 апреля", поскольку в этом месяце нет 31-го числа.

- **MONTH(<вырD>)** — определяет числовое значение номера месяца по дате <вырD>.

Пример:

```
? MONTH({05.10.91})      && 10
```

- **YEAR(<вырD>)** — определяет числовое значение года по <вырD>.

Пример:

```
? YEAR({05.10.91})      && 1991
```

Переменные и поля типа дата удобны тем, что к ним можно прибавлять и из них можно вычитать желаемое число дней и дата все равно останется корректной, т.е. будет содержать правильные новые месяц, день и год.

Примеры. Для удобства работы с датами командой SET DATE GERMAN переходим к европейскому формату даты ДД.ММ.ГГ. Создадим и исследуем переменную А типа дата, соответствующую 5 августа 1987 г.

```
.SET DATE GERMAN
.a={05.08.87}
? a+30,a-1000,a-{08.11.84}      && 04.09.87 и 08.11.84 и 1000
```

В результате выполнения действий А+30 и А-1000 получены новые даты — 4 сентября 1987 г. и 8 ноября 1984 г. Можно вычитать даты. Так, вычитание из А даты 08.11.84 снова даст 1000.

В качестве примера еще рассмотрим использование функции обработки дат при вводе данных. Пусть при вводе в переменную Т, имеющую тип дата, требуется установить диапазон разрешенных дат от 1 февраля 1988 г. до 31 декабря 1994 г. (от 01.02.88 до 31.12.94). Следующая команда позволяет удовлетворить эти ограничения:

```
@ 10,20 SAY 'Введите дату' GET t RANGE {01.02.88},{31.12.94}.
```

Пусть, кроме того, требуется не допустить, чтобы Т было нерабочим днем — субботой или воскресеньем (1- или 7-м днем недели). Для этого лучше использовать условие VALID:

```
@ 10,20 SAY 'Введите дату' GET t
VALID T>={01.02.88}.AND.T<={31.12.94}.AND.DOW(T)#1.AND.DOW(T)#7
```

Замечание. Выше были описаны несколько кажущихся бесполезными для нас функций, возвращающих английские названия элементов даты. Однако их легко можно сделать пригодными для отечественного пользователя, если несколько "улучшить" пакет FoxPro. Для этого надо просто разыскать в системных файлах FoxPro тексты этих сообщений и заменить их на русские, не изменяя дату создания файла.

Функции преобразования типов данных

- **ASC(<вырC>)** — выдает ASCII-код первого символа из <вырC>.

Пример:

```
? ASC('B')      && 66
```

- **CHR(<вырN>)** — преобразует значение <вырN> (где N — целое число от 1 до 255) в символьное, соответствующее его ASCII-коду. Функции CHR() и ASC() обратны друг другу.

Пример:

```
? CHR(66)      && B
```


■ STOD(<вырС>), DТОС(<вырD>), {<вырС>} — см. предыдущий раздел.

■ STR(<вырN>, [<длина>], [<десятичные знаки>]) — преобразует числовое <вырN> в символьную строку, включающую знак "-" (если есть) и десятичную точку, общей длиной <длина> с заданным количеством <десятичных знаков>. Если <длина> не указана, берется длина 10. Если не указаны <десятичные знаки>, число округляется до целого значения, и вообще в случае уменьшения длины дробной части оно округляется.

Пр и м е р:

.? STR(384.248,8,2)

&& 384.25

■ VAL(<вырС>) — преобразует число, представленное в символьной форме <вырС>, в числовую форму. Число разрядов дробной части получаемого числа определяется командой SET DECIMALS TO и по умолчанию равно двум. Лишние разряды округляются.

Пр и м е р:

.? VAL('-16.576')

&& -16.580

Функцией CHR() можно выдавать и непечатаемые знаки. Так, ASCII-код звукового сигнала компьютера равен 7. Тогда следующая команда не только выведет соответствующее предупреждение, но и выдаст звуковой сигнал:

.? 'ВНИМАНИЕ !!!'. CHR(7)

Используя функцию повторения, можно генерировать сигнал любой продолжительности. Например,

.? REPLICATE(CHR(7),20)

Частота и длительность сигнала могут быть установлены командой SET BELL TO.

Функции проверки файлов и дисков

Следующие функции позволяют анализировать наличие нужных файлов, текущее положение указателя в файле базы данных, успешность поиска, размеры файла, размер свободного дискового пространства и т.п.

Здесь необязательный параметр <область> означает возможность действия функции не только в активной, но и в указанной своим номером рабочей области.

■ BOF([<область>]) — достигнуто (BOF()=.T.) или нет (.F.) начало файла базы данных в текущей или указанной <области>. Функция возвращает .T. не когда указатель записей естественным образом перешел на самую первую запись базы, а в случае, если была сделана попытка выйти за ее пределы.

■ DBF([<область>]) — выдает прописными буквами полное (с указанием пути доступа) имя открытой базы в текущей или указанной рабочей <области>. Если нет, возвращается пустая строка.

■ DISKSPACE() — число свободных байт на активном диске.

■ DELETED([<область>]) — вырабатывает .T., если текущая запись помечена на удаление, и .F. в противном случае. Например, команда LIST FOR DELETED() покажет все помеченные записи.

■ EOF([<область>]) — достигнут (.T.) или нет (.F.) конец файла базы. Функция возвращает .T. не когда указатель записей естественным

образом перешел на самую последнюю запись базы, а в случае, если была сделана попытка выйти за ее пределы. При этом функция RECNO() выдает число на единицу больше фактического числа записей в базе данных.

- FILE(<имя файла>) — проверка наличия на диске файла любого типа. Имя файла (с расширением) должно быть задано в виде строки в апострофах или храниться в символьной переменной.
- FIELD(<вырN1> [,<область>]) — выдача имени поля из активной базы данных с указанным в <вырN1> номером поля базы данных по порядку. Если полей меньше, чем <вырN1>, выводится 0. Эта функция позволяет интерпретировать запись базы данных как массив с возможностью обращаться к ее полям по номеру. Имя поля выдается прописными буквами.
- FOUND([<область>]) — вырабатывает логическое значение "Истина" (.T.), если команда поиска (LOCATE, CONTINUE или SEEK) завершилась успешно, и "Ложь" в противном случае. Функция действует в текущей или любой указанной рабочей <области>. Функция удобна для анализа результатов поиска в связанных базах данных.
- FCOUNT([<область>]) — число полей в открытой базе.
- FILTER([<область>]) — возвращает прописными буквами выражение фильтра, действующего в текущей или заданной области. Функция может быть полезной при диалоговом формировании и изменении выражения фильтра отбора для базы данных.
- LOOKUP(<поле1>,<выр>,<поле2>) — ищет первое вхождение <выражения> в <поле2> активной базы и возвращает значение <поля1> из той же базы. Если файл индексируется и индекс открыт, поиск индексный, если нет, последовательный. Если поиск — неудачный, возвращается пустая строка и EOF()=.T.
- LUPDATE([<область>]) — дата последнего изменения базы данных.
- ORDER([<область>]) — выдает прописными буквами имя главного индексного файла базы, если такого индекса нет, возвращается пустая строка. Эта функция полезна для обслуживания пользовательского интерфейса, в котором возможно изменение главного индекса.
- RECNO([<область>]) — возвращает номер текущей записи активной базы данных. В качестве аргумента функции может быть задано значение 0, что имеет смысл только для индексированной базы данных. В случае, если был выполнен неудачный поиск командой/функцией SEEK, функция RECNO(0) вернет номер ближайшей, следующей за искомой записи (как если бы она была). Такое действие называется "мягким", т.е. приблизительным поиском.
- RECCOUNT([<область>]) — общее количество записей в базе данных, включая и записи, помеченные к удалению.
- RECSIZE([<область>]) — размер записи файла базы в байтах.
- HEADER([<область>]) — размер заголовка базы данных в байтах. Величина заголовка приблизительно равна $32 \cdot (N+1)$ байт, где N — число полей в базе данных.

■ **SEEK(<выр>[,<область>])** — выполняет поиск записи с <выр> в индексном файле и устанавливает на нее указатель записей. Возвращает значение .T., если поиск удачный, и .F. — если нет. Функция SEEK() эквивалентна комбинации команды SEEK и функции FOUND().

■ **UPDATE()** — вырабатывает логическое значение "Истина" (.T.) если при выполнении последней команды READ данные в ее областях GET каким-либо образом изменялись, и "Ложь" в противном случае. Использование функции, например, дает возможность не тратить время на перезапись данных на диск, если они не изменялись.

Примеры:

```

.?FILE('KADR.DBF')
.USE kadr INDEX kadrfam
.? DBF()
.? NDX(1)
.? ORDER(1)
.? BOF(),EOF(),RECNO(),RECCOUNT()
.SKIP -1
.? BOF(), RECNO(), FIELD(2)
.GO BOTTOM
.? EOF(), RECNO()
.SKIP
.? EOF(), RECNO()
.? CURDIR()
.SET FILTER TO pol='M'
.? FILTER()
.SET DEFAULT TO d:\lexicon
.? CURDIR()
.? SYS(2003)
    
```

Результаты:

```

.T.
D:\FOX2\KADR.DBF
D:\FOX2\KADRFAM.IDX
KADRFAM
.F. .F. 1 6
.T. 1 DTR
.F. 6
.T. 7
\FOX2\KADR\
POL="M"
\LEXICON\
\LEXICON
    
```

Функции BOF() и EOF() истинны только в случае, если сделана попытка выйти за пределы файла в ту или иную сторону. Если EOF()=.T., функция RECNO() получает значение на единицу больше, чем имеется записей в базе данных. Если BOF()=.T., то RECNO()=1.

По умолчанию функции DBF() и NDX() возвращают имя диска, маршрут и имя файла. Этим процессом можно управлять с помощью команды

SET FULLPATH ON/OFF

Если SET FULLPATH OFF, маршрут не возвращается (только имя диска и файла). По умолчанию — ON.

■ **CURDIR([<диск>])** — возвращает прописными буквами текущую директорию на активном или заданном <диске>. Если такого диска нет, возвращается пустая строка.

■ **SYS(5)** — возвращает имя активного по умолчанию диска. Таким диском является стартовый диск FoxPro или диск, установленный командой вида SET DEFAULT TO.

■ **SYS(2003)** — возвращает имя текущей директории на диске по умолчанию. Имя диска не выводится.

■ **SYS(2004)** — возвращает полное имя директории, где находится FoxPro.

■ **FULLPATH(<файл1> [,<вырN>/<файл2>])** — возвращает полный маршрут DOS для указанного <файла 1> или маршрут связи с другим <файлом 2>. Если имя <файла 1> задано пустой строкой (""), будет выдан полный путь к текущей директории, включая имя диска. Поиск файлов выполняется по маршруту FoxPro, т.е. маршруту, установленному командой SET PATH. <ВырN> указывает на то, что сначала будет выполняться поиск по маршруту DOS перед поиском по маршруту FoxPro. Значение <вырN> может быть произвольным.

П р и м е р. FoxPro находится в директории D:\FOXPRO2, а стартовая директория D:\PLAN. Сначала назначается директорией по умолчанию директория C:\KADR\OTD. Далее выполняется переход в директорию, где находится FoxPro, и, наконец, возврат в стартовую директорию.

```
?SYS(5), SYS(2003), SYS(2004), FULLPATH(""), CURDIR()
D: \PLAN D:\FOXPRO2\ D:\PLAN\ \PLAN\
d=SYS(5)+SYS(2003) && Сохранение маршрута стартовой директории
&& Установление новой активной директории C:\KADR\OTD
SET DEFAULT TO C:\KADR\OTD
?SYS(5), SYS(2003), SYS(2004), FULLPATH(""), CURDIR()
C: \KADR\OTD D:\FOXPRO2\ C:\KADR\OTD\ \KADR\OTD\
SET DEFAULT TO SYS(2004) && Переход в директорию СУБД
?SYS(5), SYS(2003), SYS(2004), FULLPATH(""), CURDIR()
D: \FOXPRO2 D:\FOXPRO2\ D:\FOXPRO2\ \FOXPRO2\
SET DEFAULT TO (d) && Возврат в стартовую директорию
?SYS(5), SYS(2003), SYS(2004), FULLPATH(""), CURDIR()
D: \PLAN D:\FOXPRO2\ D:\PLAN\ \PLAN\
```

Подробно ознакомиться с организацией файла базы данных можно, например, в журнале "Компьютер Пресс" N1 за 1991 г. Здесь мы лишь кратко перечислим компоненты DBF-файла. Файл состоит из заголовка и собственно данных. Заголовок содержит описатель самого файла (32 байта) и описатели каждого из полей базы (тоже по 32 байта). Далее идет байт-разделитель между заголовком и данными. Каждая запись состоит из полей плюс один байт для сохранения пометки об удалении записи (звездочки). В соответствии с этим размер заголовка базы KADR.DBF, состоящей из девяти полей, равен $(32+32*9+1) = 321$ байт. Размер записи равен сумме длин полей плюс единица — 72 байта.

С помощью функций RECCOUNT(), RECSIZE(), DISKSPACE(), HEADER() легко можно выяснить наличие достаточного места на диске при манипулировании с файлами.

П р и м е р: Найти размер базы KADR.DBF.

```
.USE Kadr
.?HEADER()+RECSIZE()*RECCOUNT()+1
```

Другой способ установить размер файла любого типа независимо от того, открыт он или закрыт, — это использовать функцию ADIR(). В нашем случае применение команды =ADIR(x,'kadr.dbf') организует массив X, где в элементе X(2) будет содержаться искомая величина (см. гл.17).

Функции позиционирования выдачи данных

Функции позволяют анализировать текущее положение курсора и головки принтера.

- COL() — номер текущей колонки на экране/окне
- ROW() — номер текущей строки на экране/окне
- PCOL() — номер текущей колонки на принтере
- PROW() — номер текущей строки на принтере

Указанные функции могут быть использованы для относительного позиционирования выдачи на экран/принтер в командах @...SAY. Например, команда

```
@ ROW()+1,5 SAY <сообщение>
```

выдаст <сообщение> на следующей строке (от текущей) в пятой колонке экрана. Вместо функций COL() и ROW() может быть использован символ \$.

Функции PCOL() и PROW() при выдаче на принтер отображают только SAY-данные и игнорируют области GET. Применение команды прогона страницы EJECT устанавливает PROW() в нуль.

Функции работы с мышью

Следующие функции дают возможность выявлять положение маркера мыши и состояние ее кнопок для создания средств управления программой.

- **MCOL([<окно>])** — возвращает колонку положения маркера мыши на экране/окне. Если <окно> не указано, выдается положение маркера относительно текущего окна/экрана. Если маркер находится вне экрана, функция возвращает -1.
- **MROW([<окно>])** — возвращает строку положения маркера мыши на экране/окне. Работает подобно функции **MCOL()**.
- **MDOWN()** — возвращает .T., если левая кнопка мыши была нажата во время выполнения функции, и .F. — если нет.

Клавишные функции

Клавишные функции позволяют создавать задержки в программе и выявлять нажатия клавиш. С помощью этих функций можно организовывать диалог пользователя в программе.

- **INKEY([<вырN>] [,<вырC>])** — выдает ASCII-код (от 0 до 255) последней клавиши, нажатой оператором на клавиатуре. Если нажатия не было, вырабатывается 0. Необязательный параметр <вырN> указывает число секунд, в течение которых ожидается нажатие клавиши для продолжения программы. Если <вырN>=0, ожидание будет неограниченным. Если <вырN> опущено, функция сработает немедленно. Параметр <вырC> управляет включением/отключением курсора и выявлением нажатия кнопки мыши. Для включения курсора в <вырC> указывается символ S, для отключения — H. Для контроля нажатия кнопки мыши используется символ M, и при этом функция будет возвращать число 151. Допускается сочетание символов M и S/H. Функция удобна для организации передачи управления внутри программы. Она может быть помещена непосредственно в WHILE-условие команды DO. Функцию можно использовать и для задания пауз установленной длины, например паузы в 20 секунд:
 = INKEY(20)

- **LASTKEY()** — возвращает ASCII-код последней нажатой клавиши. Возвращаемые коды совпадают с кодами **INKEY()**.

INKEY()/LASTKEY()-коды клавиш приведены на рис.16.1. Так, код клавиши F1 равен 28, код Shift+F1 равен 84 и т.д. При использовании этих кодов в программе обратите внимание на то, что некоторые из них совпадают.

Функции **INKEY()** и **LASTKEY()** вырабатывают одинаковые коды и кажутся очень похожими, однако это не так. Рассмотрим их подробнее. Разберем с некоторыми упрощениями такую модель ввода данных с клавиатуры (рис.16.2).

При любом вводе данные сначала заносятся в буфер клавиатуры, а после окончания ввода (обычно, но не обязательно нажатием какой-то управляющей клавиши, например Enter, Tab и др.) пересылаются в основную память компьютера. Буфер очищается, а код последней клавиши запоминается в специальной ячейке на выходе буфера. Назовем ее клавишной ячейкой. Теперь этот код может быть выявлен функцией **LASTKEY()**. Содержание клавишной ячейки остается неизменным до тех пор, пока оно не будет вытолкнуто новым вводом.

Клавиши		+Shift	+Ctrl	+Alt	Клавиши	+Shift	+Ctrl	+Alt	
F1	28	84	94	104	k	107	75	11	37
F2	-1	85	95	105	l	108	76	12	38
F3	-2	86	96	106	m	109	77	13	50
F4	-3	87	97	107	n	110	78	14	49
F5	-4	88	98	108	o	111	79	15	24
F6	-5	89	99	109	p	112	80	16	25
F7	-6	90	100	110	q	113	81	17	16
F8	-7	91	101	111	r	114	82	18	19
F9	-8	92	102	112	s	115	83	19	31
F10	-9	93	103	113	t	116	84	20	20
F11	133	135	137	139	u	117	85	21	33
F12	134	136	138	140	v	118	86	22	47
1	49	33	-	120	w	119	87	23	17
2	50	64	33	121	x	120	88	24	45
3	51	35	-	122	y	121	89	25	21
4	52	36	-	123	z	122	90	26	44
5	53	37	-	124	INS	22	48	-	-
6	54	94	30	125	HOME	1	55	29	-
7	55	38	-	126	DEL	7	46	-	-
8	56	42	-	127	END	6	49	23	-
9	57	40	-	128	PGUP	18	57	31	-
0	48	41	-	19	PGDN	3	51	30	-
a	97	65	1	30	UP	5	56	-	-
b	98	66	2	48	RIGHT	4	54	2	-
c	99	67	3	46	LEFT	19	52	26	-
d	100	68	4	32	DOWN	24	50	-	-
e	101	69	5	18	ESC	27	27	27	-
f	102	70	6	33	ENTER	13	13	10	-
g	103	71	7	34	BACKSPACE	127	127	127	-
h	104	72	8	35	TAB	9	15	148	165
i	105	73	9	23	SPACEBAR	32	32	32	32
j	106	74	10	36					

Рис.16.1

Функция `LASTKEY()` выдает код последней ФИЗИЧЕСКИ нажатой клавиши, завершающей ввод. Далее мы увидим, что код нажатой клавиши и ввод, который она осуществляет, могут и не совпадать (см. команду `SET FUNCTION`).

Функция `INKEY()` с параметром `<вырN>` является, по существу, средством ввода. Она ждет `<вырN>` секунд (если `<вырN>=0`, то без ограничения времени) нажатия любой клавиши, код которой помещается в буфер и сразу выталкивает его в клавишную ячейку.

Функция `INKEY()` вырабатывает этот код. Теперь он также становится доступным и для функции `LASTKEY()`.

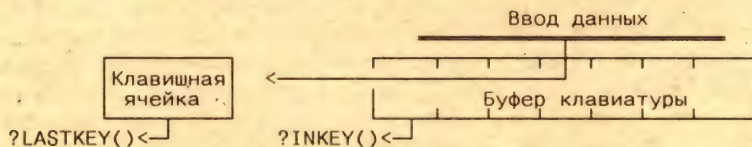


Рис.16.2

Функция `INKEY()` без параметра работает только с буфером. Если он не пуст, функция выдает код самого левого символа и выталкивает его в клавишную ячейку, подвигая все содержимое буфера влево. Повторное применение функции влечет то же действие со следующим символом, и так далее до очистки буфера. Функция удобна для последовательного анализа содержимого буфера. Поскольку такая разновидность команды `INKEY()` не предполагает клавишного ввода, буфер должен уже иметь содержимое. Это возможно осуществить командой `KEYBOARD(<вырC>)`, которая заносит в него `<вырC>`.

Следующий пример дает представление о работе функций в программе.

```
KEYBOARD 'ABC'
?INKEY(), LASTKEY(), INKEY(), LASTKEY(), INKEY(), INKEY()
65      65      66      66      67      0
```

Здесь числа 65, 66, 67 являются кодами букв А, В, С.

- **READKEY()** — выдает код клавиши, нажатой для выхода из команды редактирования READ. Нажатой клавише соответствуют два кода. Один — если данные в команде не изменялись, другой (увеличенный на 256) — если изменялись. На рис.16.3 приведены важные для нас клавиши покидания окна редактирования из любого места. Можно также покинуть окно, любым образом выведя курсор за последнее/первое GET-поле.

Клавиша	READKEY-коды
PgUp	6/262
PgDn	7/263
Esc	12/268
F1	36/292
Ctrl-Home	33/289
Ctrl-PgUp.	34/290
Ctrl-PgDn	35/291
Ctrl-End	/270

Рис.16.3

В числителе указан код, соответствующий выходу без изменения данных, в знаменателе — с изменениями.

Кроме того, функция получила новые возможности для организации взаимодействия с командой READ по управлению интерфейсом. Здесь эти опции не рассматриваются.

- **CAPSLOCK([<вырL>])** — переключает клавишу CapsLock управления регистром символов (ПРОПИСНЫЕ/строчные буквы). CAPSLOCK(.T.) включает, а CAPSLOCK(.F.) выключает. Если <вырL> опущено, состояние режима не меняется, а функция выдает его текущее значение (.T./F.). Функция удобна в тех случаях, когда необходимо гарантировать ввод прописными или строчными буквами.

П р и м е р:

```
. =CAPSLOCK(.T.)
.?CAPSLOCK()                                && .T.
```

- **CHRSAW([<вырN>])** — возвращает .T. через <вырN> секунд, если в буфере клавиатуры имеется символ. Если <вырN> опущено, функция немедленно возвращает значение. В отличие от функции INKEY() содержание буфера при этом не изменяется. Функция удобна для анализа факта ввода с клавиатуры в течение заданного времени.

- **ON(<вырC1>[,<вырC2>])** — возвращает закрепленную за событием (нажатием клавиши, ошибкой) команду. <ВырC1> — название одной из команд вида ON ERROR, ON ESCAPE, ON KEY, ON PAGE, ON READERROR. <ВырC2> — имя клавиши, если имеется в виду команда ON KEY LABEL.

П р и м е р:

```
. ON KEY LABEL Ctrl+C CANCEL
.ON('KEY','Ctrl+C')                                && CANCEL
```

- **VARREAD()/SYS(18)** — эти полностью идентичные функции возвращают (прописными буквами) имя поля/переменной, при вводе

данных в которое командой READ было выполнено прерывание от клавиши с помощью процедуры ON KEY ... или сделан вызов пользовательской функции в опциях VALID/WHEN. Функции удобны, например, для вызова экранов подсказок при вводе данных, для каждого поля — своей подсказки. Функции можно использовать и с командой BROWSE/CHANGE. Значение SYS(18)/VARREAD() может быть передано в пользовательские функции, вызываемые в опциях :W, :V, :E или командой

ON KEY LABEL <клавиша> DO <процедура> WITH VARREAD()

Однако имя поля передается в них, но не прописными, а (кроме первой буквы) строчными буквами. При желании для перехода к прописным буквам можно воспользоваться функцией UPPER()).

- SYS(2002 [,1]) — включает, выключает (с параметром 1) курсор на экране терминала. Аналогичное действие вызывает команда SET CURSOR ON/OFF.

Технические функции

Технические функции позволяют выяснить характеристики и состояние устройств конкретного компьютера и текущей среды прикладной системы для настройки прикладных программ, а также установить правильную последовательность индексирования.

- FKMAX() — возвращает число фактически доступных функциональных клавиш компьютера. Например, на клавиатуре их может быть 12, а доступны лишь 10.
- SET(<вырC>[,1]) — текущий статус различных SET-команд. Здесь учитывается и то, что некоторые такие команды имеют две разновидности. Если параметр 1 опущен, выдается статус команды вида SET ... ON/OFF, если нет — вида SET SET ... TO. Функция может быть очень полезной для анализа текущего состояния прикладной системы. Например, если ранее была выдана команда SET ALTERNATE TO plan назначения альтернативного текстового файла с именем PLAN.TXT, то применение функций SET() даст следующие результаты:

```
.?SET('alter')                && OFF
.?SET('alter',1)              && D:\FOX2\PLAN.TXT
```

Здесь сделан запрос о последней команде SET ALTERNATE ON/OFF и имени файла, назначенного к выводу текстовых данных. Параметр выдается целиком, включая имя диска и путь доступа. В качестве аргумента функции допускается задавать и 4-буквенные сокращения.

- SYS(2006) — типы платы адаптера и монитора компьютера.

Пример:

```
.? SYS(2006)                  && EGA/Color
```

- SYS(17) — тип используемого процессора (i8086/88, i80286, i80386, i80486).
- SYS(13) — возвращает прописными буквами статус готовности принтера — "Готов" (READY), "Нет" (OFFLINE). Функция может использоваться для организации взаимодействия пользователя с программой при необходимости печати данных.
- PRINTSTATUS() — возвращает логическое .T., если принтер готов к

работе, и .F. — если нет. Функция похожа на SYS(13), но удобнее.

Пример:

```
IF !PRINTSTATUS()
    ? "Подготовьте принтер!"
ENDIF
```

- **SYS(12)** — число байт памяти, свободных для использования (в пределах 640 Кбайт), что полезно при конфигурировании FoxPro.
- **MEMORY()** — размер доступной памяти для загрузки внешних программ по команде RUN/! в виде числового значения в килобайтах. Похожа на SYS(12), которая выдает строку и в байтах.
- **SYS(1001)** — размер всей имеющейся памяти компьютера (до 1 Мбайта).
- **SYS(1016)** — память, в текущий момент занятая под временные переменные, окна, меню, открытые файлы.
- **ISCOLOR()** — возвращает .T., если компьютер имеет цветной монитор и .F. — если нет. Функция полезна для настройки готовой прикладной системы на имеющийся (цветной или монохромный) монитор.
- **SCHEME(<вырN1>[,<вырN2>])**. — возвращает цветовую пару номер <вырN2> списка цветовых пар из схемы цветов номер <вырN1>. Если <вырN2> опущено, выдается весь список. Функция полезна при отладке, если вы хотите выяснить, каким цветом раскрашен данный элемент/элементы экранного объекта.
- **SYS(15,<вырC1>,<вырC2>)**
Функция может быть нужна для пользователей, работающих с национальным алфавитом, если его буквы расположены в кодировочной таблице не в алфавитном порядке. Функция SYS(15) заменяет каждый символ строки <вырC2> на соответствующий (по номеру) символ из <вырC1>. Если в <вырC1> не найден символ, соответствующий символу из <вырC2>, этот символ не будет заменен. К этому вопросу мы еще вернемся.
- **VERSION([1])** — номер версии пакета FoxPro, а также (если указан параметр 1) дата создания и номер самого пакета.

Функции времени

Функции времени необходимы для ведения баз данных, где нужно, например, фиксировать текущее время ввода данных. Кроме того, эти функции полезны программисту для выяснения скорости работы тех или иных фрагментов программы. Другое их полезное применение наряду с функциями, вырабатывающими даты, — это использование для назначения имен файлов (например, текстовых), создаваемых в программе. Поскольку давать имена файлам на русском языке нельзя, можно назначать осмысленные для пользователя имена, содержащие хотя бы дату и время их создания.

- **TIME()** — возвращает системное время в виде строки, состоящей из восьми символов в двадцатичетырехчасовом формате (ЧЧ:ММ:СС).
- **SYS(2)** — возвращает в виде строки символов число секунд с полуночи.
- **SECONDS()** — возвращает системное время в секундах. Функция удобна для анализа временных характеристик программ, а также для установления абсолютного промежутка времени между некоторыми

событиями внутри текущего сеанса работы с системой, например для установления факта двойного/тройного нажатия мыши.

Пр и м е р:

```
.? SECONDS()
```

```
&& 71798.533
```

Функция анализа условий

- **IIF(<условие>,<выр1>,<выр2>)** — выдает значение <выр1>, если <условие> истинно, и <выр2> — если ложно. Функция удобна для применения в командах индексирования, в команде BROWSE, при печати, при описании меню и т.д., поскольку ее можно непосредственно включать в команды (например, ?, @...SAY и др.). Вообще, функция IIF() может быть помещена практически вместо любого <выражения>, присутствующего в команде или другой функции. Важно только, чтобы тип этого выражения соответствовал разрешенному в данном месте.

Пр и м е р. Пусть требуется из базы KADR.DBF вывести поле DET (число детей) таким образом, что, если детей нет, выводится не цифра 0, а слово HET. Следующие команды это реализуют:

```
.USE kadr
.LIST FIELDS fam,dtr,pol,IIF(det=0,'HET',det) OFF
```

Допускается вложение функций IIF. Положим, при выводе данных из файла нужно указать пол сотрудника не буквами "М" или "Ж", а словами "Мужчина" или "Женщина" и семейное положение не буквами "Б", "Х", "Р", а словами "В браке", "Холост", "Разведен".

```
.LIST fam,IIF(pol='М','Мужчина','Женщина'),
      IIF(sem='Б','В браке',IIF(sem='Х','Холост','Разведен'))
```

Функции анализа типа и наличия данных

- **EMPTY(<выр>)** — возвращает логическое .Т., если объект <выр> (поле/переменная) не имеет данных, и .F. в противном случае. В зависимости от типа данных "пустым" считается поле/переменная со значением, указанным на рис.16.4.

Тип данных	Содержание
Символьный	Пробелы, знаки табуляции, возвраты каретки, переводы строк
Цифровой	0
Даты	Нулевое значение { . . . } или { }
Логический	.F.
Мемо-поле	"Пустой" (отсутствует содержимое мемо-поля)

Рис.16.4

- **TYPE(<вырС>)** — возвращает прописной буквой символ, обозначающий тип данных для <вырС>: С — символьный, L — логический, N — числовой, М — мемо-поле, D — дата, U — неопределенный. Имя анализируемой переменной должно быть взято в апострофы/кавычки.

Пр и м е р:

```
.USE kadr
.?TYPE('dtr'),TYPE('i=2')
```

```
&& D, L
```

Финансовые функции

Хотя финансовые функции присутствуют практически во всех крупных

пакетах обработки данных (СУБД, электронные таблицы), ранее они мало интересовали отечественных пользователей. Очевидно, что теперь они будут все шире применяться в повседневной практике.

Все функции этой категории связаны с размером кредита/вклада, процентными ставками и платежами/выплатами. Далее предполагается, что процентная ставка постоянна и выражена в десятичной форме и что платежи вносятся в конце фиксированных периодов времени. Кроме того, значения всех величин внутри функции должны быть приведены к одинаковому расчетному периоду (году, месяцу и т.д.).

■ **PAYMENT(<вырN1>,<вырN2>,<вырN3>)**

Функция вычисляет размер периодических <выплат> за взятый <кредит>, на который установлен определенный <процент>. Известно также <число периодов>, за которое кредит должен быть погашен. Иными словами:

$$\langle \text{выплата} \rangle = \text{PAYMENT}(\langle \text{кредит} \rangle, \langle \text{процент} \rangle, \langle \text{число периодов} \rangle)$$

Процентная ставка должна соответствовать периоду, через который делаются платежи. Поскольку обычно известен годовой процент, то он должен быть пересчитан в зависимости от промежутка между платежами.

П р и м е р. Пусть в банке взят на два года кредит в размере 100 000 руб. из расчета 10 % годовых (0.1/12 месячных) с выплатой ежемесячно (за 24 месяца).

$$? \text{ PAYMENT}(100000, .1/12, 24) \quad \&\& 4614.49$$

Таким образом, чтобы с учетом процентов погасить кредит, требуется 4 614.49 руб. ежемесячных взносов или

$$? \text{ PAYMENT}(100000, .1, 2) \quad \&\& 56719.05$$

два ежегодных взноса по 56 719.05 руб.

Функция осуществляет вычисления по формуле $N1 \cdot N2 / (1 - 1 / (1 + N2)^{N3})$, что в нашем случае даст

$$? 100000 \cdot .1 / (1 - 1 / (1 + .1)^{24}) \quad \&\& 56719.05$$

Следующая функция определяет растущий вклад клиента на счету банка в зависимости от выплачиваемого банком процента, размера периодического взноса и числа вкладов (числа периодов).

■ **FV(<вырN1>,<вырN2>,<вырN3>)**

что соответствует выражению

$$\langle \text{вклад} \rangle = \text{FV}(\langle \text{периодический взнос} \rangle, \langle \text{процент} \rangle, \langle \text{число периодов} \rangle)$$

П р и м е р:

$$? \text{ FV}(1000, .1/12, 36) \quad \&\& 41781.82$$

Таким образом, при ежемесячных взносах в размере 1000 руб. в течение трех лет и 10 % годовых будет накоплено 41 781.82 руб. Функция реализует вычисления по такой формуле $N1 \cdot ((1 + N2)^{N3} - 1) / N2$, что подтверждается следующим результатом:

$$? 1000 \cdot ((1 + .1/12)^{36} - 1) / (.1/12) \quad \&\& 41781.82$$

Если функция FV() вычисляет будущее значение вклада, то следующая функция определяет современное значение суммы, которая будет получена в будущем.

■ **PV(<вырN1>,<вырN2>,<вырN3>)**

Или, иными словами,

$\langle \text{выплата} \rangle = PV(\langle \text{периодические выплаты} \rangle, \langle \text{процент} \rangle, \langle \text{число периодов} \rangle)$

П р и м е р. Пусть за аренду помещения банк гарантирует вам ежегодные выплаты в размере 10 000 руб. в течение трех лет, т.е. всего 30 000 руб. Но выплата такой суммы в течение трех лет с учетом растущих процентов соответствует некоторой меньшей сумме, пересчитанной на текущий день. Установим ее. Положим также, что процентная ставка составляет 10 % годовых. Если выплаты будут производиться ежемесячно по 10000/12 руб., то

? PV(10000/12; .1/12,36)

&& 25826.03

То есть общая сумма в 30 000 руб. превращается в 25 826.03 руб. на сегодняшний день. Это и есть текущая стоимость ренты.

Ниже приведены рабочая формула и результаты расчета по ней:

$$N1 * (1 - (1 / (1 + N2) ** N3)) / N2$$

? 10000/12 * (1 - (1 / (1 + .1/12) ** 36)) / (.1/12)

&& 25826.03

Механизм пересчета значений будущих выплат на сегодняшний день называется дисконтированием (учетом векселей). Обычное применение функции PV() на Западе — это расчет расходования пенсионером накопленного им пенсионного фонда.

Функции подстановки

Функции подстановки позволяют интерпретировать символьную переменную как отображаемый ею объект программы. Они дают возможность формировать имена переменных, функции и даже целые команды непосредственно в программе.

■ & <символьная переменная> [. <вырC>]

Функция называется функцией макроподстановки и превращает содержимое <символьной переменной> или мемо-поля непосредственно в объект, который она изображает.

Например, открытие файла BRIG1 может быть выполнено с использованием макроподстановки следующим образом:

```
.a='BRIG1'
.USE &a
```

Это эквивалентно команде USE BRIG1 и в данном случае не дает никаких преимуществ. Однако, если имя файла должно быть сформировано программным образом, макроподстановка незаменима.

Пусть на диске имеется столько файлов бригадных выработок, сколько бригад в цехе. Имена этих файлов состоят из двух компонентов — фиксированного (слова BRIG) и изменяющегося (номера бригады): BRIG1, BRIG2, BRIG3, ...

В следующем фрагменте программы пользователь вводит номер нужной бригады, по которому открывается соответствующий файл BRIG:

```
.INPUT 'Укажите номер бригады' TO nom
.n='brig'+LTRIM(STR(nom))
.USE &n
```

Здесь номер бригады вводится в цифровой форме в переменную NOM. Затем в переменной N формируется полное символьное имя файла как соединение слова BRIG и номера бригады. Причем сначала NOM преобразуется в символьную строку функцией STR(), а затем из нее

удаляются ведущие пробелы функцией LTRIM(). Если пробелы не удалить, то, например, для NOM=3 значение N будет равно 'BRIG 3', а не 'BRIG3'.

Нежелательные пробелы возникают из-за того, что целые части числовых переменных имеют в FoxPro длину в 10 разрядов (фактически NOM=0000000003) и ведущие нули функцией STR() превращаются в пробелы.

Можно вводить номер бригады в символьной форме, например командами @ ... GET, АСЦЕПТ или иным образом. В этом случае нет необходимости использовать функцию STR(). Что касается удаления ведущих пробелов, то это нужно делать всегда, когда они мешают. В рассмотренном примере пробелы не могут входить в имена файлов.

Аргументом функции макроподстановки является все то, что следует от знака & до первого пробела. Если необходимо составить фразу, содержащую кроме самой макроподстановки некоторую строку <вырС>, их следует соединить точкой. Если <вырС> само начинается с точки, то их должно быть две.

Пр и м е р. Положим, что нужно удалить один из бригадных файлов с диска. В команде удаления ERASE имя файла должно включать и расширение, в нашем случае DBF:

```
.INPUT 'Укажите номер удаляемой бригады -' TO nom
.n='brig'+LTRIM(STR(nom))
.ERASE &n..dbf
```

В случае формирования команд для работы с именами (файлов, переменных, полей) допускается использовать также символьные переменные, взятые в круглые скобки. Применительно к предыдущему примеру аналогичное, но более быстрое действие даст команда

```
.ERASE (n+'..dbf')
```

Макроподстановка может применяться не только для формирования имен, но и целых команд. Например, тот же самый результат, что и в предыдущем случае может быть получен таким образом:

```
.INPUT 'Укажите номер удаляемой бригады -' TO nom
.n='ERASE brig'+LTRIM(STR(nom))+'.dbf'
.&n
```

Однако не допускается

```
. &'ERASE brig'+LTRIM(STR(nom))+'.dbf'
```

поскольку после знака & может стоять только символьная ПЕРЕМЕННАЯ. Возможны вложения макроподстановок друг в друга. Общая длина одной макроподстановки после выполнения преобразований не должна превышать 255 знаков. В случае, если это невозможно, макроподстановка разбивается на части. Например разобьем команду DISPLAY ALL следующим образом: X='DISPLAY' и Y='ALL'. Тогда можно записать команду из двух макроподстановок — &X &Y. Пробел между ними обязателен, поскольку пробел имеется в этом месте и в команде DISPLAY ALL. Напротив, если X='DISP' и Y='LAY ALL', то только — &X&Y.

Макроподстановка, содержащая FOR-условие может выполняться с привлечением оптимизирующей технологии поиска (технология Rushmore).

Следующая функция

■ EVALUATE(<вырС>)

рассматривает <вырС> как некоторое допустимое для FoxPro выражение и возвращает вычисленный ею результат. В отличие от макроподстановки (&) возможности функции скромнее, и она может использоваться только в составе команд, но выполняется быстрее.

Пример:

```
.? EVALUATE('2**3'),EVALUATE('DATE()')  
.USE kadr  
.? EVALUATE('fam')
```

&& 8, 09-03-92

&& СИДОРОВ П.С.

Функции подстановки являются исключительно полезным инструментом построения программ.

Вместе с функциями в этом разделе уместно рассмотреть одну команду. Функции могут применяться только в составе команд. Однако часто бывает нужно использовать функцию, следствием применения которой является некоторое действие, результат которого нельзя или не нужно присваивать никакой переменной.

Есть команда, осуществляющая вызов функции/функций без присвоения:

■ = <выр>[,<выр>] ...

В качестве <выражений> можно использовать как встроенные, так и пользовательские функции. Например, команда

```
.= CAPSLOCK(.t.)
```

осуществляет переход на верхний регистр клавиатуры, а команда

```
.= INKEY(120)
```

создает паузу в две минуты и при этом код нажатой клавиши (если нажатие было сделано) не будет отображен на экране.

Кроме встроенных функций, рассмотренных выше, программист может организовывать и собственные функции в виде процедур.

Глава 17. МАССИВЫ ПЕРЕМЕННЫХ

Одним из важных инструментов программиста является возможность работы с массивами временных переменных.

Описание массивов

В FoxPro разрешается работа с одномерными и двумерными массивами переменных. Для этого они предварительно должны быть описаны специальной командой

- DECLARE/DIMENSION <переменная> (<вырN1> [, <вырN2>])
[, <переменная> (<вырN1> [, <вырN2>])] ...

Нумерация элементов массива начинается с единицы.

Н а п р и м е р:

```
DIMENSION a(3,8), b(4)
```

Здесь описываются два массива: двумерный массив А (размерностью 3х8) и одномерный В (длиной четыре элемента). Сразу после описания массивов командой DIMENSION все их элементы получают логический тип со значением .F. В дальнейшем элементы в результате присваиваний могут получать новые типы и значения. Тип каждого отдельного элемента определяется результатом последней команды присваивания (=, STORE, SCATTER).

При описании массива и обращении к его элементам разрешено использовать круглые или квадратные скобки. Разрешается повторное описание существующего массива, причем имеющиеся данные не будут утрачены. Это может иметь смысл, если после заполнения массива оказалось, что ранее было зарезервировано больше элементов, чем необходимо.

Каждый массив использует для своего имени одну переменную и может иметь до 3600 элементов (по умолчанию). Разрешено использовать до 3600 массивов. Практические ограничения зависят от доступной памяти компьютера.

Обмен данными с базой данных

Обмен данными между массивами и базами данных может выполняться посредством двух пар команд, которые реализуют считывание записей в массив и занесение массива в записи базы данных.

Работа с одномерными массивами/переменными. Следующая команда последовательно переносит значения полей только из текущей записи активного файла базы данных в последовательные элементы одномерного массива (если указана опция <массив>) или переменные (если указано MEMVAR). Все такие переменные получают те же типы и размеры, что и поля базы данных.

- SCATTER [FIELDS <поля>] [MEMO]
TO <массив> [BLANK] / [MEMVAR] [BLANK]

Если массив ранее не был описан или его длина недостаточна, он будет создан командой. Если переменные ранее не существовали, они будут созданы с теми же именами, что и соответствующие поля. Для того чтобы СУБД могла отличить одинаковые имена полей и переменных, к последним нужно обращаться с префиксом "М". Например, М->Х или М.Х для переменной Х.

Если использовано слово BLANK, то создается множество незаполненных переменных или элементов массива, однако имеющих те же типы и размеры, что и поля базы данных.

Если режим FIELDS не указан, переносятся все поля записи.

Опция MEMO указывает на то, что и мемо-поля будут копироваться. При этом необходимо следить, чтобы размер копируемого мемо-поля не был больше разрешенной длины переменной.

П р и м е р. В результате использования следующих команд будут созданы массив A размерностью три элемента и переменные M.FAM и M.TAB:

```
USE kadr
SCATTER FIELDS fam,tab,per TO a MEMO
SCATTER FIELDS fam,tab MEMVAR
```

Действие команды GATHER обратно действию команды SCATTER:

■ GATHER FROM <массив>/MEMVAR [FIELDS <список полей>] [MEMO]

Команда переписывает в <поля> текущей записи активного файла базы данных элементы <массива> или одноименные переменные (опция MEMVAR), созданные ранее командой SCATTER, включая и мемо-поля (если указана опция MEMO).

Типы соответствующих полей и элементов массива должны совпадать. Если отсутствует слово FIELDS, элементы переносятся в последовательные поля, начиная с первого.

Если число элементов массива и полей не совпадает, лишние игнорируются.

Рассмотренная пара команд очень удобна, если предстоит сложная обработка полей записи базы данных. В этом случае лучше перенести их в переменные, а затем, после обработки, вернуть в базу. Часто ввод/редактирование данных приходится сочетать с некоторой их трансформацией, сложными проверками и сопутствующими вычислениями, например данные из текущей записи могут понадобиться для просмотра других записей в этой же базе.

Приведем фрагмент программы копирования полей базы KADR.DBF во временные переменные для редактирования с возвратом их назад:

```
USE kadr
SCATTER MEMVAR MEMO
@...SAY 'Фамилия' GET m.fam
@...
READ
GATHER MEMVAR MEMO
```

Работа с двумерным массивом. Следующая пара команд осуществляет взаимодействие с двумерными массивами. Мемо-поля командами игнорируются. Команда

■ COPY TO ARRAY <массив> [FIELDS <поля>] [<граница>] [FOR <вырL1>] [WHILE <вырL2>]

последовательно пересылает поля из записей текущей базы данных в последовательные элементы строк <массива>.

Команда COPY TO в отличие от команды SCATTER не создает массива, т.е. он к этому времени должен существовать. Причем такой массив должен быть описан как двумерный, даже если считывается одно поле (например, A(10,1), но не A(10)). Если количество элементов массива и полей не совпадает, то лишние игнорируются.

Команды COPY TO ARRAY и SCATTER близки по функциям, но команда COPY позволяет переслать в массив несколько записей, а SCATTER — только одну, и команда COPY сама не создает массива.

Если <массив> двумерный, действие команды заканчивается после заполнения всех строк массива или достижения конца базы.

Следующая команда:

■ APPEND FROM ARRAY <массив> [FOR <вырL>] [FIELDS <поля>]

по своему действию обратна действию команды COPY TO ARRAY и добавляет к базе данных записи из <массива> так, что каждая строка массива становится записью базы.

Если массив имеет больше элементов, чем база данных полей, то лишние элементы отбрасываются, и, наоборот, если база имеет больше полей, лишние поля остаются пустыми. Если указано FOR-условие, добавление новой записи из массива произойдет только в случае, если оно истинно. <ВырL> должно содержать имя поля/полей базы данных, но проверяются элемент/элементы массива, которые направляются в это поле. Если условие ложно, строка массива пропускается. Если <массив> двумерный, действие команды заканчивается после просмотра всех строк массива.

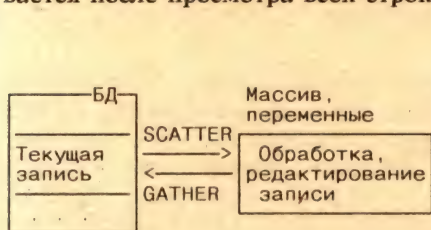


Рис.17.1

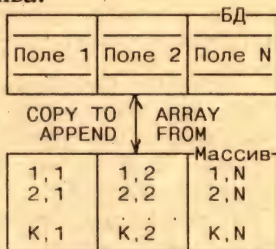


Рис.17.2

Кроме рассмотренных команд в системе имеется большая группа функций по обработке именно массивов (поиск, сортировка, удаление и включение данных). Хотя формально это функции, по существу они являются командами. Обычный способ их применения — включение в команду "=".

Возможны два способа указания элемента в двумерном массиве — указание обоих его индексов, т.е. номеров строки и столбца, либо указание его порядкового номера в массиве. Оба вида нумерации показаны на рис.17.3.

(1, 1) или 1	(1, 2) или 2	(1, 3) или 3
(2, 1) или 4	(2, 2) или 5	(2, 3) или 6
(3, 1) или 7	(3, 2) или 8	(3, 3) или 9

Рис.17.3

Функции AELEMENT() и ASUBSCRIPT() позволяют выяснить местонахождение элемента, заданное любым способом.

Функция

■ ASUBSCRIPT(<массив>,<вырN>,<1/2>)

возвращает из двумерного массива положение его элемента номер <вырN>. Если третий аргумент равен 1, возвращается номер строки, если равен 2 — номер столбца.

Функция

■ AELEMENT(<массив>,<вырN1>[,<вырN2>])

возвращает одномерный номер элемента двумерного <массива> по номеру строки <вырN1> и столбца <вырN2>. Номер L элемента A(I,J) массива размерности A(K,N) вычисляется по формуле $L=(I-1)*N+J$. Функция AELEMENT() является обратной к функции ASUBSCRIPT().

Пример:

DIMENSION a(5,7)
? AELEMENT(a,3,2), ASUBSCRIPT(a,16,1) && 16 и 3

Функция

■ ADEL(<массив>, <вырN> [,2])

удаляет один элемент одномерного <массива> или строку/колонок двумерного <массива>. Если удаление выполнено успешно, функция возвращает 1. Размерность такого массива не изменяется. Удаляемые элементы получают значения .F. и становятся в конец массива. Их место замещается ближайшими элементами. Если удаляется строка/столбец, то их место замещается следующими строками/столбцами. <ВырN> указывает номер удаляемого элемента одномерного массива или строку/колонок двумерного массива. Включение параметра [,2] означает, что удаляется не строка, а столбец.

Функция

■ AINS(<массив>,<вырN> [,2])

вставляет один элемент одномерного или строку/колонок двумерного <массива>. Если вставка выполнена успешно, функция возвращает 1. Размерность такого массива не изменяется, а новые элементы получают значения .F.. Элементы/строки/столбцы, на место которых производится вставка, отодвигаются назад/вниз/вправо. Самые крайние компоненты массива оказываются за его пределами и утрачиваются. <ВырN> указывает номер вставляемого элемента одномерного массива или строку двумерного массива. Включение параметра [,2] означает, что вставляется не строка, а столбец.

Функция

■ ACOPY(<массив1>,<массив2> [,<вырN1>[,<вырN2>[,<вырN3>]])

копирует элементы <массива1> в <массив2>. Если <массив2> не существует, он создается и его размерность будет той же, что и размерность <массива1>. Функция возвращает количество скопированных элементов.

Здесь:

<вырN1> — элемент в исходном <массиве1>, с которого начинается копирование в <массив2>;

<вырN2> — число копируемых элементов;

<вырN3> — элемент <массива2>, с которого начинается ввод копируемых данных.

Функция

■ ASCAN(<массив>,<выр> [,<вырN1>[,<вырN2>]])

ищет <выражение> среди элементов <массива>, начиная с элемента номер <вырN1> до элемента номер <вырN2>. Возвращается номер найденного элемента или 0 в противном случае.

Функция

■ ASORT(<массив>[,<вырN1>[,<вырN2>[,<вырN3>]])

сортирует элементы <массива>. Для одномерного массива сортировка выполняется, начиная с элемента номер <вырN1> до элемента с номером <вырN2>. Для двумерного массива сортировка заключается в перестановке строк без изменения их содержимого. Сортировка массива выполняется по столбцу номер <вырN1>, в соответствии с которым переставляются <вырN2> строк (по умолчанию все строки). <ВырN3> может быть равно 0/1 и определяет вид сортировки — возрастание/убывание. Если не задан ни один из необязательных параметров, массив сортируется целиком.

П р и м е р. Пусть имеется массив A(3,2), над которым последовательно

выполняются функции ASORT(a,1) и ASORT(a,2,2). Слева направо изображены исходный массив и его новое содержимое после применения функций.

8	1		3	9		5	4
5	4	->	5	4	->	3	9
3	9		8	1		8	1

Здесь сначала весь массив сортируется по возрастанию элементов первого столбца, а затем, также по возрастанию, но элементов второго столбца и только его двух первых строк.

Функция

■ AFIELDS(<массив>)

позволяет получить из открытой базы данных сведения о ее структуре, проанализировать структуру и, возможно, создать новую базу командой CREATE DBF (из языка SQL). В результате применения функции будет создан двумерный массив из четырех столбцов, содержащих имя поля, его тип (одна буква), размерность и число десятичных знаков соответственно. Число строк будет совпадать с числом полей базы. Используя перечисленные ранее функции ADEL(), AINS(), ACOPY(), можно легко перестроить массив и создать на его основе новую базу.

Следующая последовательность команд:

```
USE kadr
= AFIELDS(a)
LIST MEMORY LIKE a
```

позволит создать такой массив из базы KADR.DBF и просмотреть его на экране.

Функция

■ ADIR(<массив>[,<вырC1>])

заполняет <массив> информацией о файлах, возможно, в соответствии с маской (<вырC1>) и возвращает количество найденных файлов.

Функция создает двумерный <массив> с числом строк, равным числу выявленных файлов, и пятью столбцами. Столбцы содержат информацию DOS о каждом из файлов — имя файла, размер, дату, время создания и атрибут файла соответственно. Типы элементов строки — символьный, числовой, дата, символьный, символьный.

П р и м е р. Создать массив BD, содержащий сведения о всех базах с именами, начинающимися с буквы К.

```
=ADIR(bd, 'k*.dbf')
```

Используя функцию ADIR(), можно легко находить размеры любых файлов и суммарное дисковое пространство, занимаемое ими (элемент массива номер (1,2)). Такая возможность важна при копировании и преобразовании данных для контроля величины необходимого дискового пространства. Другое применение функции ADIR() — это создание массива для построения меню из имен файлов, если стандартные возможности POPUP-меню с опцией PROMPT FILES для нас оказываются недостаточными.

Глава 18. ОРГАНИЗАЦИЯ МЕНЮ В ПРИКЛАДНЫХ СИСТЕМАХ

Меню является основной формой диалога в прикладных системах обработки данных. FoxPro обладает исключительно развитыми средствами поддержания меню как с объемным световым курсором (световым "зайчиком"), так и с назначаемыми клавишами. Для удобства будем называть их световым и клавишным меню соответственно.

Кроме того, в FoxPro реализовано еще и "кнопочное" (Button) меню. Термин "кнопка" здесь означает не физическую кнопку на клавиатуре, а некоторую область на экране, которой приданы управляющие свойства. "Кнопочные" меню удобно использовать совместно с мышью. Такие меню рассмотрены в гл.25, 26, посвященных организации интерфейса в стиле Windows.

18.1. Световое меню

Световое меню представляет собой перечень элементов меню, среди которых один текущий "подсвечен" (отображен иным цветом, чем другие пункты меню). Перемещение светового курсора осуществляется клавишами управления курсором. Выбор в меню производится нажатием клавиш Enter и Space (Ввод и Пробел), отказ от выбора — клавишей Escape.

Большинство из реализованных в языке меню является, по существу, окнами, т. е. они могут проектироваться на видимую часть экрана без "порчи" изображения под ним. При деактивации меню экран автоматически восстанавливается. Это исключительно удобно при работе, например, с экранами редактирования данных. Здесь можно в любой момент нажатием некоторой предварительно закрепленной клавиши вызвать меню на экран, сделать необходимый выбор и вернуться назад.

Все виды меню допускают работу с мышью, а также выбор по первой букве элемента меню. При этом, если установлена команда SET CONFIRM OFF, нажатие клавиши с первой буквой немедленно повлечет и выбор данного пункта. Если установлена команда SET CONFIRM ON, то потребуется подтверждение выбора нажатием клавиши Enter/Space.

Когда некоторые пункты меню начинаются на одну и ту же букву, это может быть неудобно. В таком случае можно назначить свои "горячие" клавиши. Если в строку меню включены символы "\<", то символ, стоящий справа от них, будет выделен цветом. Нажимая клавишу с этим символом, мы можем также сделать выбор в меню, причем без нажатия Enter независимо от команды SET CONFIRM.

Если пункт меню начинается с символа "\", он будет отображаться на экране приглушенным цветом и не будет выбираться (курсор на нем не фиксируется). Если пункт (только в POPUP-меню) состоит из двух символов "\-", то ему в меню будет соответствовать горизонтальная линия. Использование таких элементов позволяет зрительно группировать пункты меню, разделяя их горизонтальными линиями ("\-"), и даже давать им свои подзаголовки ("\").

В FoxPro имеются две альтернативные технологии построения меню — концепция, продолжающаяся из предыдущей версии СУБД FoxBASE-2.1, и концепция, позаимствованная из СУБД dBASEIII.

Прежде чем перейти к изучению собственно технологии построения и использования меню, уместно указать различия Fox- и dBASE-меню (назовем их так). Важнейшее "идейное" различие между ними может быть определено терминами "меню-программа" и "меню-объект". Fox-меню, как мы и привыкли, является только частью программы, где оно создается, используется и

"умирает". dBASE-меню после своего определения остается независимым и "живым" объектом, к которому можно обратиться из любого места прикладной системы, и даже в командном окне уже после завершения программы.

Фох-меню всегда вырабатывают числовые переменные, фиксирующие сделанный пользователем выбор. Эти переменные далее анализируются в программе, обычно в структуре DO CASE ... ENDCASE. Для поддержания возможности постоянного возврата в меню оно или команда активации меню, как правило, помещается в цикл вида DO WHILE .T. ... ENDDO.

dBASE-меню может не только вырабатывать переменные для анализа, но и непосредственно вызывать процедуры, подпрограммы, команды по обработке выбора. Оно наряду с традиционной (реализованной в Фох-меню) предлагает совершенно иную архитектуру построения систем обработки данных, когда программа может вообще не иметь единого ядра, а вся состоять из процедур, которые связаны друг с другом только через вызовы меню. Кроме того, имеется неопределимая возможность создавать меню из имен файлов и компонентов базы данных, осуществлять множественный выбор.

На практике можно успешно пользоваться всеми формами меню, сочетая их удобным образом. Однако, даже если вы (скорее всего) уже привыкли к такой организации меню, как это принято в Фох-меню и вообще в алгоритмических языках, и не хотите отказываться от предыдущего опыта, все равно найдите возможность изучить средства dBASE-меню. Потраченное время будет в дальнейшем возмещено, и вы, скорее всего, совсем перейдете на новый стиль программирования меню-программ. Сама фирма Fox Software отдает предпочтение dBASE-меню. Кроме того, Генератор меню FoxPro использует именно эту форму.

Сначала, как более простое, рассмотрим Фох-меню. Однако читатель может и опустить этот раздел, сразу обратившись к более "перспективному" dBASE-меню.

18.1.1. FOX-меню

В FOX-меню возможно создание трех различных форм меню:

1. Меню с произвольно расположенными элементами (меню с подсветкой элементов — LIGHTBAR-меню).
2. Вертикальное ("всплывающее" POPUP-меню).
3. Двухуровневое меню (PULLDOWN-меню).

Названия меню в скобках соответствуют названиям в технической документации.

Меню первого типа одновременно определяется и используется. В меню второго и третьего типов процесс определения формы и содержания меню отделен от его использования, что более удобно.

LIGHTBAR-меню

Для создания такого меню используются команды @ ... PROMPT, MENU и, возможно, SET MESSAGE. Рассмотрим их подробнее.

Команда

■ @ Y,X PROMPT <вырC1> [MESSAGE <вырC2>]

выдает в позиции Y,X строку меню (<вырC1>). Информативность меню может быть усилена включением в команду фразы MESSAGE <вырC2>. Тогда в последней или определенной командой SET MESSAGE строке экрана возникнет дополнительное сообщение (<вырC2>).

Само меню состоит из ряда (до 128) команд PROMPT, предъявляющих в

заданных позициях экрана строки меню. Перемещение внутри меню пользователь осуществляет нажатием клавиш со стрелками, причем активная в данный момент строка меню выделяется контрастным изображением. Выбор необходимой позиции меню производится нажатием клавиши Enter или Space.

К аналогичным результатам ведет нажатие клавиши, соответствующей самому первому символу из необходимой позиции меню. Для этого удобно, например, пронумеровать все пункты меню.

Команда

■ MENU TO <переменная>

осуществляет запоминание цифры, соответствующей порядковому номеру выбранной строки PROMPT в некоторой переменной. Здесь имеется в виду номер команды в тексте программы (среди других команд PROMPT, относящихся к данному меню), а не номер ее позиции на экране. Нажатие клавиши Escape также влечет выход из меню с занесением нуля в <переменную>.

Команда

■ SET MESSAGE TO [<вырN> [LEFT/RIGHT/CENTER]]

применяется, чтобы в строке <вырN> окна/экрана вывести дополнительное сообщение к пункту меню, на котором стоит курсор, слева, справа или в центре строки (LEFT, RIGHT или CENTER). В таком меню должна быть опция MESSAGE с <вырC>. Команда SET MESSAGE TO без параметра <вырN> направляет сообщение в последнюю строку экрана.

Команда действует как в Fox-меню, так и в dBASE-меню, но в последнем случае MESSAGE-сообщение центрируется автоматически.

Пр и м е р. Построим меню с пунктами "Конец работы", "Дополнение данных", "Корректировка", "Печать данных" и выдачей дополнительных сообщений "Включите принтер" при выборе позиции меню "Печать данных" и "Выход из системы" при выборе позиции "Конец работы".

```
CLEAR
SET MESSAGE TO 16 CENTER
@ 5,31 SAY 'МЕНЮ СИСТЕМЫ:'
@ 7,31 PROMPT 'Конец работы' MESSAGE 'Выход из системы'
@ 9,31 PROMPT 'Дополнение данных'
@ 11,31 PROMPT 'Корректировка'
@ 13,31 PROMPT 'Печать данных' MESSAGE 'Включите принтер'
MENU TO R
SET MESSAGE TO
```

В примере дополнительные сообщения будут выведены в центре 16-й строки (SET MESSAGE TO 16 CENTER).

При выборе пользователем строки КОНЕЦ РАБОТЫ переменная R получит значение 1, при выборе строки ДОПОЛНЕНИЕ ДАННЫХ — 2 и т.д.

Такое меню можно расположить не только вертикально, но и горизонтально и как угодно. Более того, можно запрограммировать динамическое меню (с изменяющимися числом строк и их содержанием), ведь сообщения в строке PROMPT, т.е. <вырC1>, могут быть и переменными.

Вертикальное POPUP-меню

Формирование и использование такого меню существенно упрощены по сравнению с предыдущим. При этом разделены процессы его наполнения (в специальных символьных массивах) и применения. Это делает процесс программирования более удобным, а программу более обозримой.

Для обширных меню, не уместяющихся на экране, возможна их прокрутка нажатием клавиш управления курсором. Такое меню создается командами @ ... MENU и READ MENU TO.

Следующая команда описывает форму и содержание меню:

■ @ Y,X MENU <массив>,<вырN1> [,<вырN2>] [TITLE <вырC>]

Здесь:

Y,X — координаты левого верхнего угла окна меню на экране;

<массив> — имя одномерного массива, который содержит элементы меню символьного типа длиной до 50 знаков;

<вырN1> — число элементов меню (до 128);

<вырN2> — число одновременно изображаемых на экране элементов меню (до 17). По умолчанию представляется 17 элементов или все меню (если в нем не более 17 строк);

TITLE <вырC> — заголовок меню, изображенный в его верхней части.

Команда @ ... MENU только отображает меню на экране.

Активация меню осуществляется командой

■ READ MENU TO <переменная> [SAVE]

Здесь <переменная> — переменная числового типа, которая фиксирует номер позиции меню, выбранной пользователем. Если <переменная> перед включением в команду определена, ее значение укажет начальное положение курсора в меню. Если нет или это значение находится за пределами <массива>, курсор установится на первую строку меню.

Параметр SAVE позволяет сохранить меню на экране после его использования. Далее оно может быть стерто командой CLEAR.

Выявление выбора пользователя в программе осуществляется путем анализа <переменной>, обычно в структуре DO CASE ... ENDCASE.

П р и м е р. Приведем программу создания трехстрочного меню из массива A(3) и меню-переменной M, а также само меню. При его активации курсор окажется в позиции РЕДАКТИРОВАНИЕ (m=2).

```
CLEAR
DIMENSION a(3)
a(1)=' ВВОД ДАННЫХ'
a(2)=' РЕДАКТИРОВАНИЕ'
a(3)=' УДАЛЕНИЕ'
m=2
@ 8,30 MENU a,3 TITLE 'МЕНЮ'
```

МЕНЮ
ВВОД ДАННЫХ
РЕДАКТИРОВАНИЕ
УДАЛЕНИЕ

```
READ MENU TO m
```

П р и м е р. Решим задачу множественного выбора с помощью меню, одновременно рассмотрев работу с массивами.

Пусть имеется символьный массив A длиной 10 элементов, т.е. A(10), содержащий определенные фамилии. Нужно сформировать другой массив B(10), в который пользователь должен занести выбранные им произвольно из A(10) некоторые фамилии.

Ниже приводится программа, реализующая нашу задачу.

```
SET SAFETY OFF
SET TALK OFF
DIMENSION b(10)
CLEAR
IF FILE('aa.mem')
    RESTORE FROM aa ADDITIVE    && Если есть файл AA.MEM,
    DIMENSION a(10)             && считывание массива из файла ELSE
                                &&-----Формирование массива A-----
```



```

a=SPACE(16)
@ 5,0
FOR i=1 TO 10
  @ $+1,30 SAY 'a('+LTRIM(STR(i))+')' GET a(i)
ENDFOR
READ
SAVE TO aa ALL a          && Сохранение массива A в файле
ENDIF
@ 6,10 MENU a,10 TITLE 'Пробел - конец'
CLEAR                    &&-----Отбор в массив B-----
j=0                      && Текущий номер элемента в B
@ 4,16 SAY 'Массив A: => Массив B:'
@ 5,10 SAY REPLICATE('-',45)
x=1
DO WHILE .T.
  READ MENU TO x SAVE    && Активация меню
  DO CASE
    CASE x=0              && Если нажата клавиша Escape,
      CANCEL              && отбор отменен
    CASE LASTKEY()=32     && Если нажата клавиша Space,
      EXIT                && конец отбора
    OTHERWISE             && Иначе
      j=j+1               && приращение номера элемента,
      b(j)=a(x)           && перенесение выбора в массив B,
      @ 6+x,40 SAY b(j)   && показ выбранной строки справа,
      a(x)='\' +a(x)      && скрытие отработанной строки меню
    ENDCASE
  ENDDO
  bj=j                    && Запоминание отобранного количества
  @ 20,29 SAY 'ОТОБРАНО '+LTRIM(STR(j))
  SAVE TO bb ALL LIKE b* && Сохранение массива B в файле BB.MEM

```

Вначале объявляются массивы A(10) и B(10). Чтобы главная задача могла быть выполнена, необходим уже заполненный массив A. Однако поскольку мы предполагаем, что читатель захочет опробовать программу и тогда потребуются заполненный массив, в нее внесены команды по вводу данных в A(10).

Для хранения временных переменных нам понадобятся два файла типа MEM. Назовем их AA для массива A и BB для массива B. Сначала анализируется наличие на диске файла AA.MEM.

Если файл есть, он считывается с диска в основную память (RESTORE FROM aa ADDITIVE). Затем осуществляется переход к главной задаче выбора. Режим ADDITIVE указан, чтобы не потерять объявленный и поэтому существующий, хотя и пустой, массив B.

Если файла нет (ELSE), выполняется ввод данных в A. Предварительно всем элементам массива A под фамилии присваивается значение пустого слова длиной 16 (SPACE(16)).

В FOR-цикле, начиная с пятой строки, выводятся на экран (@ \$+1,30 SAY...GET...) имена элементов массива ("A(1)", "A(2)", ... "A(10)") и области для ввода данных.

После завершения цикла на экране будут предъявлены все десять, пока пустых, элементов массива A, которые пользователь может заполнить по своему усмотрению. При этом имеется постоянная возможность редактирования уже введенных элементов, пока не будет заполнен последний десятый элемент массива или совершен выход из редактирования, например по PgDn/PgUp.

Собственно присвоение изображенных на экране данных из областей GET элементам массива A осуществляется командой READ. Сформированный массив запоминается в файле AA.MEM (SAVE TO aa ALL a).

Далее устанавливается POPUP-меню. В заголовок меню помещаем указание на возможность прекращения отбора нажатием клавиши Пробел. Выводится заголовок экрана 'Массив A: => Массив B:', подчеркнутый двойной линией, и в цикле активизируется меню (READ MENU TO x).

Во второй, главной, половине программы пользователю в левой части

экрана предлагается меню из всех десяти фамилий массива А. Передвигая курсор и нажимая клавишу Enter, можно выбирать желаемые фамилии, которые переносятся в массив В. Для самоконтроля выбора все эти фамилии тут же отображаются справа от соответствующих элементов массива А. Закончить отбор можно, нажав клавишу Space. В результате выполнения программы получаем заполненный массив В и число отобранных в него фамилий в переменной ВJ. Число отобранных людей индицируется на экране.

Возможен также отказ от выбора нажатием клавиши Escape. Вид экрана на момент окончания отбора представлен на рис.18.1.

Массив А:	=> Массив В:
(Пробел-конец)	ЛУКИН А.П.
ЛУКИН С.Н.	ВАСИН А.Н.
КРУГЛОВА В.П.	
ВАСИН А.Н.	
.....	
ОТОБРАНО 2	

Рис.18.1

В рассматриваемой части программы переменная J — счетчик элементов, отобранных в В, X — номер выбранной позиции меню (номер элемента в А).

После того как пользователь нажал какую-либо чувствительную клавишу в меню, он его покидает, а нажатие анализируется. Если X=0, значит, нажата клавиша Escape, и программа заканчивается (CANCEL). Если код нажатой клавиши 32 (пробел) — завершается отбор: осуществляется выход из цикла, выдается сообщение об отобранном числе фамилий и в файле BV.MEM записываются все выбранные фамилии вместе с их числом. Для того чтобы запомнить число отобранных фамилий, переменная J пересылается в переменную ВJ. Благодаря этому легко указать общую маску (В*) имен переменных В и ВJ для сохранения их в файле.

Если нажата клавиша Enter, значит, в меню выбрана фамилия, которая пересылается в массив В и отображается справа от меню. Чтобы избежать повторного выбора одной и той же фамилии, использованные строки меню подавляются введением перед фамилией знака "\".

Далее в цикле опять будет предложено то же самое меню с возможностью сделать очередной выбор. Программу можно легко модифицировать для любого количества элементов массива А.

Двухуровневое PULLDOWN-меню

Такое меню состоит из главного горизонтального меню (меню заголовков), располагающегося в верхней части экрана, и нескольких вложенных в него вспомогательных вертикальных меню. Каждое вспомогательное меню возникает на экране в тот момент, когда курсор перемещается в соответствующую позицию главного меню. Выбор нужного пункта меню, как и ранее, делается нажатием клавиш Enter/Space.

Структура двухуровневого меню реализуется двумя командами определения меню: MENU BAR, MENU и командой активации READ MENU BAR TO. Рассмотрим их.

Определение главного горизонтального BAR-меню выполняется командой

■ MENU BAR <массив>,<вырN>

Здесь:

<вырN> — общее число пунктов меню;

<массив> — двумерный символьный массив вида МАССИВ(K,2).

Первый столбец массива (элементы МАССИВ(i,1)) содержит собственно пункты меню, второй столбец (МАССИВ(i,2)) — расширенные комментарии к позициям меню, индицируемые в строке, номер которой определяется командой SET MESSAGE TO <номер строки>.

Если эти комментарии не нужны, массив все равно должен быть двумерный и элементы МАССИВ(i,2) должны быть символьного типа, хотя бы и нулевой длины.

На экране предъявляется столько пунктов меню, сколько удастся разместить в строке экрана. Остальные позиции делаются доступными путем прокручивания.

Команда

■ MENU <вырN1>,<массив>,<вырN2>[,<вырN3>]

определяет содержание вспомогательного POPUP-меню и его "привязку" к соответствующему пункту главного меню.

Здесь:

<вырN1> — номер элемента главного меню, который будет вызывать данное вспомогательное меню;

<массив> — одномерный символьный массив, содержащий пункты меню;

<вырN2> — число пунктов меню. Обычно, но не обязательно, оно равно размерности <массива>;

<вырN3> — число одновременно показываемых пунктов меню. Если оно меньше <вырN2>, то меню прокручивается. Параметр используется, если для размещения меню не хватает строк экрана.

Активирует меню команда

■ READ MENU BAR TO <перем1>,<перем2> [SAVE]

Где <переменные 1 и 2> — переменные числового типа, которые фиксируют выбор пользователя в меню: <перем1> — номер пункта главного меню, <перем2> — номер пункта вспомогательного меню. Если выбор в меню не сделан (нажата клавиша Escape), обе переменные получают значение 0. Их исходные значения (если есть) определяют начальное положение курсора в меню, если эти значения находятся за пределами <массивов> или отсутствуют, курсор установится в положение 1,1.

Включение параметра SAVE означает, что использованное меню будет не стерто, а сохранено на экране.

Двухуровневое меню формируется в следующей последовательности. Сначала, обычно в начале модуля, описываются и заполняются массивы главного и всех вспомогательных меню, а также инсталлируются главное (командой MENU BAR) и вспомогательные меню (командами MENU).

В нужных местах программы остается только указать команду READ MENU BAR TO, которая активирует само меню.

Далее идут команды обработки выбора в меню, обычно включаемые в структуру DO CASE ... ENDCASE.

На рис.18.2 приводится пример такого меню. Здесь для наглядности показаны сразу все три вспомогательных вертикальных меню, хотя на экране, конечно, можно одновременно видеть только одно из них. Кроме собственно меню в 20-й строке экрана в зависимости от положения курсора в главном меню появляются дополнительные сообщения из массива A (элементы A(1,2)).

Пункту горизонтального меню КОНЕЦ не соответствует, никакое вспомогательное меню. Это значит, что при выборе указанного пункта <переменные 1 и 2> получают значения $G=1$ и $V=0$ соответственно.

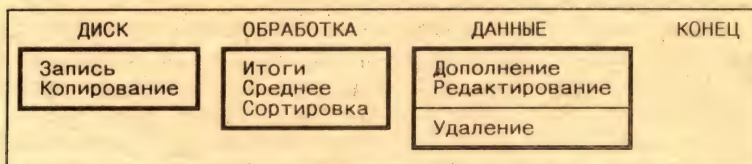


Рис.18.2

```

SET TALK OFF
SET MESSAGE TO 20
DIMENSION a(4,2)
a(1,1)=' ДИСК
a(2,1)=' ОБРАБОТКА
a(3,1)=' ДАННЫЕ
a(4,1)=' КОНЕЦ
a(1,2)=' РАБОТА С ДИСКОМ
a(2,2)=' ВЫЧИСЛЕНИЯ И ПОИСК
a(3,2)=' РАБОТА С БАЗОЙ
a(4,2)=' ВЫХОД В ДОС
DIMENSION b(2)
b(1)='запись'
b(2)='копирование'
DIMENSION c(3)
c(1)='итоги'
c(2)='среднее'
c(3)='сортировка'
DIMENSION d(5)
d(1)='дополнение'
d(2)='редактирование'
d(3)='\'
d(4)='удаление'
* Формирование образа меню
MENU BAR a,4
MENU 1,b,2
MENU 2,c,3
MENU 3,d,4
g=3
v=2
DO WHILE .T.
  READ MENU BAR TO g,v
  DO CASE
    <команды обработки выбора>
    CASE g=4
      CANCEL
    ENDCASE
  ENDDO

```

Как видим, для меню, предполагающих неоднократное обращение (а таких большинство), в программе необходима организация цикла по его регенерации. Здесь мы использовали цикл DO WHILE ... ENDDO с всегда истинным условием .T.. Циклы неудобны, поскольку путают программиста, в особенности если создается сложная иерархическая система меню. Указанного недостатка лишено dBASE-меню. При своем определении оно остается независимым объектом и может быть вызвано в любом месте программы.

После обработки выбора из такого меню обеспечивается автоматический в него возврат. Кроме того, dBASE-меню предлагает ряд новых важных функций, о которых говорилось выше.

18.1.2. dBASE-меню

В общем случае для создания dBASE-меню и работы с ним программисту необходимо предусмотреть следующие элементы:

- Определение меню. Здесь описываются содержание, "горячие" клавиши, клавиши быстрого доступа, форма и реакции меню. Определение меню может быть сделано один раз в начале программы и далее только использоваться. Разделение процессов определения меню и его активации очень удобно и делает программу более "прозрачной".
- Активация меню. Команды/клавиши активации предъявляют меню на экране и делают его чувствительным к выбору пользователя.
- Деактивация меню удаляет его с экрана, сохраняя в памяти для следующего возможного использования.
- Удаление меню. Это действие очищает память от определения меню, и более оно не может быть использовано без нового определения. Конечно, удобно, когда меню постоянно находится в памяти и всегда доступно. Однако ограничивающим фактором здесь является память компьютера, и поэтому иногда приходится прибегать к временному удалению меню с повторным, если необходимо, его определением. В таком случае определение даже удобно поместить в отдельную процедуру.

dBASE-меню имеет два типа элементарных меню:

1. Вертикальное ("всплывающее", или POPUP-меню),
2. Горизонтальное (BAR-меню).

На их основе можно строить иерархические меню практически любой сложности (вложенности). Технология создания любого типа меню включает перечисленные в начале раздела этапы.

Этот материал очень важен для понимания принципов построения меню, и вместе с тем он, возможно, покажется запутанным и слишком насыщенным новыми понятиями. Чтобы облегчить его осознание, введем несколько терминов, опираясь на названия команд.

BAR-меню — это обычно горизонтальное меню (BAR — по-английски строка, линейка). Такое меню имеет имя, данное ему по правилам, принятым в FoxPro, и состоит из конкретных элементов, пунктов, которые будем называть PAD-пунктами (PAD — заголовок). Каждый PAD-пункт также имеет имя и видимую на экране строку-приглашение.

POPUP-меню — это прямоугольное меню, строки-элементы которого будем называть BAR-пунктами или BAR-строками (не путать с BAR-меню). POPUP-меню имеет имя, а его видимое содержание на экране (BAR-строки) будет зависеть от типа POPUP-меню.

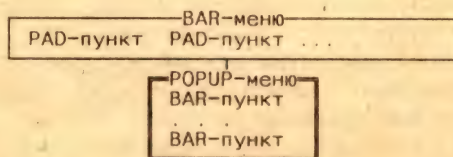


Рис.18.3

BAR- и POPUP-меню могут использоваться отдельно, а могут быть связаны, как показано на рис.18.3.

Прежде чем перейти к изучению конкретных команд, рассмотрим некоторые общие для них опции, с тем чтобы не останавливаться на них дальше:

COLOR SCHEME <вырN>/**COLOR** <список цветовых пар> — установление цвета элементов меню. По умолчанию используется цветовая схема номер 2.

MESSAGE <вырC> — возможное дополнительное сообщение к меню, возникающее в центре нижней строки, или в строке, указанной командой **SET MESSAGE**. В иерархических меню **MESSAGE**-сообщения нижнего уровня имеют приоритет перед сообщениями верхнего уровня. Так например, если опция **MESSAGE** была использована и при определении **PAD**-пункта и для подчиненного ему **POPUP**-меню, то выведено будет <вырC> для последнего.

IN [WINDOW] <окно>/IN SCREEN — указывает, где будет показано меню. Это может быть <окно> или экран (**SCREEN**). По умолчанию оно предъявляется в текущем окне/экране.

KEY <имя клавиши>[, <вырC>] — указывает <имя клавиши>, которая может быть использована для вызова меню. Эта опция равнозначна (см. разд. 18.2 "Клавишное меню") команде

ON KEY LABEL <имя клавиши> **ACTIVATE POPUP** <POPUP-меню> для вызова **POPUP**-меню. Такое назначенное <имя клавиши> отображается справа от соответствующего пункта меню. Если имя клавиши оказывается слишком длинным для отображения на экране (в строке меню мало места), его можно заменить на другое, указав в <вырC>. Например, имя **Ctrl+End** можно заменить на более короткое слово "End" (т.е. **KEY Ctrl+End, "End"**).

MARK <вырC> — устанавливает символ, который будет показан слева от выбранного пункта меню. По умолчанию это ромб (**ASCII**-код 4). Этим процессом можно управлять также с помощью команды **SET MARK OF** (см. документацию).

SKIP [FOR <вырL>] — пункт меню показывается на экране приглушенным цветом, и его выбор невозможен. Если указан параметр **FOR <вырL>**, пункт меню будет доступен только при <вырL>=.F.. Опция позволяет блокировать бессмысленные или неразрешенные в данный момент пункты меню.

NOWAIT — эта опция может включаться в команды активации меню. Меню, активированное таким образом, не останавливает выполнения программы, которое продолжается со следующей команды. Использование этой возможности целесообразно, если ниже имеются объекты, создающие состояние ожидания, например **READ**- и **BROWSE**-окна. Обычно такое меню как раз и создается для их "обслуживания".

Вертикальное POPUP-меню

"Всплывающее" **POPUP**-меню предъявляется в виде прямоугольника в указанном месте окна/экрана. Содержанием меню могут быть имена файлов, компоненты файлов или произвольные строки. Меню допускает "прокручивание", а также выбор по первой (для всех видов меню) или назначенной букве строки меню (исключая меню из компонентов файлов). Здесь используются следующие средства.

Определение меню. Командой описания меню **DEFINE POPUP** дается имя меню, указывается его местонахождение, цвет и содержание (если используются параметры **PROMPT FIELD/FILES/STRUCTURE**), а также назначаются "горячие" клавиши выбора пунктов внутри меню и клавиша

вызова самого меню. Если содержание меню состоит из произвольных строк, оно не может быть определено только командой DEFINE POPUP. В этом случае элементы меню персонально описываются командами DEFINE BAR.

Командой ON SELECTION POPUP определяется реакция на выбор из меню нажатием клавиши Enter/Space. Можно также назначить действия на выбор отдельных строк меню, в том числе вызов других меню с помощью команд ON [SELECTION] BAR.

Активация меню. Меню, описанное указанными выше командами, может быть вызвано в любом месте программы командой ACTIVATE POPUP или KEY-клавишей. Теперь с ним может работать пользователь. Если в программе нужно уточнить выбор пользователя, применяются специальные меню-функции (BAR(), POPUP(), ...).

Деактивация/удаление меню. Меню может быть отключено командой DEACTIVATE POPUP, или нажатием клавиши Escape, или клавишами с горизонтальными стрелками. Если необходимость в POPUP-меню отпала совсем, лучше его удалить из памяти командами CLEAR POPUPS или RELEASE POPUPS. Если его нужно только временно удалить с экрана, используется команда HIDE POPUP.

Рассмотрим команды организации меню.

Описание POPUP-меню. Команда описывает прямоугольное меню и определяет его содержание, если оно является компонентами файла/файлов.

```
■ DEFINE POPUP <POPUP-меню>
  [FROM <Y1>,<X1>] [TO <Y2>,<X2>]
  [PROMPT FIELD <выр>/
    PROMPT FILES [LIKE <маска>]/
    PROMPT STRUCTURE]
  [IN [WINDOW] <окно>/IN SCREEN]
  [FOOTER <вырC1>] [KEY <имя клавиши>]
  [MARGIN] [MARK <вырC2>]
  [MULTI] [MESSAGE <вырC3>]
  [SCROLL] [TITLE <вырC4>] [SHADOW]
  [COLOR SCHEME <вырN>/COLOR <список цветовых пар>]
```

Здесь:

<POPUP-меню> — имя POPUP-меню, которое дает программист.

FROM <Y1,X1> TO <Y2,X2> — координаты левого верхнего и правого нижнего углов POPUP-меню в окне/экране. Если параметры Y2, X2 не заданы, ширина меню будет определена по максимальному элементу, а высота — во весь экран или по числу элементов меню (если их меньше). Если отсутствуют первые координаты меню, то Y1=0 и X1=0 или же оно располагается непосредственно под соответствующим PAD-пунктом более старшего BAR-меню (если есть).

PROMPT FIELD <выр> — элементами меню будут записи из открытой базы данных, содержащие заданное поле-выражение. <Выражение> может содержать и не одно только поле, а несколько, соединенных знаком "+", в том числе и из других открытых баз. В этом случае они должны быть приведены к одному обычно символьному типу. Допускается использовать и функции, в том числе ПФ. Это удобно, если мы хотим видеть не просто перечень выбираемых объектов, но и результаты какого-то их анализа. Разрешается использовать в меню и поля из базы

данных, находящейся в другой рабочей области. Тогда имя поля должно быть составным — с включением имени области. Предельный размер базы, которая может быть вызвана в POPUP-меню, 32 767 записей.

Так следующее меню TABEL выводит в координатах 5x8 и 18x30 поле TAB из базы данных KADR.DBF

```
USE kadr
DEFINE POPUP tabel FROM 5,8 TO 18,30 PROMPT FIELD kadr.tab
```

PROMPT FILES [LIKE <маска>] — элементами будут названия файлов, возможно, ограниченные <маской>.

Например, следующая команда создаст описание меню с именем TEXT, которое отобразит имена всех файлов с расширением TXT. Вызываться такое меню может нажатием клавиш Alt-5.

```
DEFINE POPUP text PROMPT FILES LIKE *.txt KEY ALT+5
```

PROMPT STRUCTURE — в качестве элементов меню будет предъявлена структура открытой базы данных. Команды

```
USE kadr
DEFINE POPUP kadrstr PROMPT STRUCTURE
```

определяют меню KADRSTR, представляющее структуру базы KADR.DBF.

Ниже показаны связи PROMPT-опций команды с наполнением меню

```
DEFINE POPUP<> PROMPT — { FIELD<>   - меню из записей БД
                           FILES<>    - меню из файлов
                           STRUCTURE   - меню из структуры БД
```

FOOTER <вырC1> и TITLE <вырC4> — заголовки меню, располагаемые в центре нижней и верхней границ области меню.

MULTI — устанавливает режим множественного отбора из меню. Когда пункт меню выбран, слева появляется специальный знак (по умолчанию ромб). Если выбран еще другой пункт, то знак переносится туда. Если вы хотите пометить (выбрать) сразу несколько пунктов меню, то после того, как выбран какой-нибудь один пункт и курсор передвинут к другому, следует, удерживая клавишу Shift, нажать Enter/Space. Можно пометить сразу несколько смежных пунктов меню, если, нажав Shift на помеченной строке, двигать курсор с помощью клавиш со стрелками. Можно пометить даже все строки меню, начиная от текущей, вверх/вниз, если нажать в этом случае клавишу PgUp/PgDn (помечаются все видимые строки), Home/End (все строки до начала/конца меню). Далее с помощью функции MRKBAR() можно будет определить, какие именно пункты меню были отобраны.

Этот режим не работает в POPUP-меню, которые используют файлы (опции PROMPT FIELD/FILES/STRUCTURE).

MARGIN — устанавливает пробелы справа и слева от приглашений меню. Они могут понадобиться для вывода символа пометки строки меню и/или маркера текущего положения курсора в меню (опции SCROLL и MULTI). В противном случае указанные символы будут проектироваться прямо на строки меню, затерев последний/первый символ строки меню.

SCROLL — если пункты меню не умещаются в окне/экране, справа появляется вертикальная полоса, на которой будет показан маркер текущего положения курсора. Это позволяет пользователю видеть свое положение среди данных, а также удобно для работы с мышью.

SHADOW — предъявление меню сопровождается "тенью".

Пример. Пусть требуется построить меню из полей FAM (фамилия), DTR (дата рождения) и DET (число детей) базы KADR.DBF. Причем из даты рождения нужен только год рождения, а фамилию в целях экономии места нужно отобразить 12 символами. Но главное — это установить кандидатов на материальную помощь. Пусть помощь назначается из расчета 400 руб. на одного ребенка, но не более 1000 руб. Вычисленное число также должно отображаться в меню. Программа, реализующая поставленную задачу, и вид меню (рис.18.4) приведены ниже.

```
.USE kadr
.DEFINE POPUP kadr FROM 1,1 TO 5,40;
  TITLE 'Фамилия | Год рожд. | Детей | Помощь';
  PROMPT FIELD LEFT(fam,12)+' | '+STR(YEAR(dtr),4)+' | '+'
  STR(det,2)+' | '+' STR(MIN(det*400,1000),4)
.ACTIVATE POPUP kadr
```

Фамилия	Год рожд.	Детей	Помощь
РОМАНОВА. М.С	1966	0	0
ПОТАПОВ Д.П	1960	3	1000
ЯКОВЛЕВ А.И.	1930	2	800

Рис.18.4

Теперь, например, можно выбрать кандидатов, которым необходима материальная помощь.

Замечание к постановке задачи. Здесь производится именно выбор кандидатов на помощь с учетом некоторых возможных обстоятельств, которые не отражены в базе данных, например личные данные, срок и размер последней помощи и т.д. Если бы помощь назначалась автоматически по числу детей, то было бы достаточно одной команды LIST:

```
LIST FOR det#0 fam, MIN(det*400,1000) OFF
```

Используя функции в выражении для выводимых полей меню, можно построить простой фильтр данных, видимых в меню. Например, команда вида

```
DEFINE POPUP kadrм PROMPT FIELD IIF(pol='М',fam,SPACE(25))
```

описывает меню KADRM, которое предъявляет фамилии только мужчин. Фамилии женщин замещаются на пустые строки. Однако, поскольку для таких строк остается возможность выбора, то в самом начале процедуры обработки выбора следует предусмотреть анализ фактического значения поля POL, либо значения функции PROMPT(). Например, если EMPTY(PROMPT()), обработка выбора запрещается.

Отбор в меню только нужных записей может быть сделан с помощью команды SET FILTER TO либо созданием и активацией отдельного индекса с FOR-условием.

Описание элементов POPUP-меню. Команда описывает BAR-пункты POPUP-меню, если программист задает их сам в виде некоторых строк. Команда используется после команды DEFINE POPUP.

- DEFINE BAR <вырN1> OF <POPUP-меню> PROMPT <вырC1>
 [KEY <имя клавиши>[,<вырC2>]] [MARK <вырC3>]
 [MESSAGE <вырC4>] [SKIP [FOR <вырL>]]
 [COLOR <список цветовых пар>/COLOR SCHEME <вырN2>]

Здесь:

DEFINE BAR <вырN1> — номер описываемого в команде BAR-пункта меню.
OF <POPUP-меню> — имя POPUP-меню, которому принадлежит этот пункт.
PROMPT <вырC1> — содержание BAR-пункта на экране (приглашение).

Строка <вырC1> может включать и функции, что позволяет создавать меню, где приглашение будет выглядеть различным образом в зависимости от обстоятельств. Например, пусть рабочая область В отведена для бригадных файлов (файлов вида BRIG<номер>.DBF). Тогда пункт меню описанный следующим образом:

```
... PROMPT IIF(EMPTY(DBF('b')), 'Бригады нет', ;
  'Бригада номер '+LEFT(RIGHT(DBF('b'),5),1));
  SKIP FOR EMPTY(DBF('b'))
```

предъявит строку вида 'Бригада номер <номер>', если открыта какая-либо база бригад в области 'b', и строку 'Бригады нет', если ни один из бригадных файлов не открыт. Одновременно эта строка меню будет заблокирована.

По умолчанию строки меню, описанные командами DEFINE BAR <номер> OF, располагаются в соответствии со своими номерами. Если же какой-нибудь номер отсутствует, он заменяется пустой строкой, на которой курсор не останавливается.

Определение реакции меню. Следующие команды указывают, что произойдет, если выбор в меню сделан.

■ ON SELECTION POPUP <POPUP-меню>/ALL [<команда>]

Здесь задается имя POPUP-меню, выбор из которого фиксируется, а также <команда>, которая выполняется при нажатии клавиши Enter. Хотя это может быть <команда> любого типа, обычно это команда DO, вызывающая процедуру. Такая процедура после своего завершения снова возвращает нас к POPUP-меню, если конечно, в ней не встретилась команда DEACTIVATE POPUP, которая обеспечит выход из POPUP-меню на команду, следующую за командой ACTIVATE POPUP.

Параметр ALL указывает на то, что команда будет выполняться не для одного какого-то POPUP-меню, а для всех активированных на данный момент.

Следующие команды при необходимости позволяют назначить пунктам POPUP-меню индивидуальные реакции (команды, процедуры, другие меню).

■ ON SELECTION BAR <вырN> OF <POPUP-меню> [<команда>]

Назначает <команду> (обычно вызов процедуры) на пункт номер <вырN> из названного <POPUP-меню>. Назначение должно быть сделано после определения <POPUP-меню>, но до его активации. Использование команды без параметра <команда> отменяет закрепление.

Команда

■ ON BAR <вырN> OF <POPUP-меню1> [ACTIVATE POPUP <POPUP-меню2>/ACTIVATE MENU <BAR-меню>]

назначает вызов другого POPUP/BAR-меню на пункт номер <вырN> из названного <POPUP-меню1>. Использование команды без обязательного параметра отменяет закрепление.

Если за данным пунктом закреплен вызов другого меню, то справа от

строки приглашения будет показана стрелка-треугольник. Чтобы для нее было выделено место в прямоугольнике меню, следует определять меню, из которого происходит вызов, с опцией MARGIN.

Активация меню. Команда предъявляет меню на экране и активирует его

- **ACTIVATE POPUP** <имя POPUP-меню>
[AT <Y,X>] [BAR <вырN>] [NOWAIT] [REST]

После активации пользователь может делать выбор из меню.

Опции команды:

AT <Y,X> — определяет координаты экрана/окна для вывода меню. Эти координаты имеют преимущество над координатами, указанными при описании меню в команде **DEFINE POPUP**.

BAR <вырN> — задает номер текущей строки меню при его выводе.

REST — текущей строкой меню будет текущая запись базы данных. По умолчанию текущей строкой будет первая запись. Имеет смысл для меню, описанного с опцией **PROMPT FIELD**.

Функции выбора из меню

Чтобы запомнить выбор в меню, сделанный пользователем, в процедуру можно передать параметры выбора с помощью следующих функций:

- **BAR()** — функция возвращает номер выбранного BAR-пункта POPUP-меню, определенный командой **DEFINE BAR**. Возвращается 0, если выбор не был сделан (например, нажата клавиша Escape). Такой номер присваивается командой **DEFINE BAR <вырN>...**
- **POPUP()** — возвращает заглавными буквами имя активного POPUP-меню.
- **PROMPT()** — возвращает строку-приглашение, выбранную в POPUP-меню. Если команда **DEFINE POPUP** использовалась с фразой **FIELD**, функция возвратит содержимое поля из выбранной записи базы данных; если со словом **FILE** — имя файла с расширением и маршрутом доступа; если со словом **STRUCTURE** — имя выбранного поля; если выбор не сделан — пустую строку.
- **CNTBAR(<POPUP-меню>)** — возвращает число BAR-строк в <POPUP-меню>. Функция может быть полезной, если меню имеет переменное число строк.
- **MRKBAR(<POPUP-меню>,<вырN>)** — возвращает значение .T., если BAR-строка номер <вырN> из <POPUP-меню> помечена (выбрана), и .F. в противном случае. Функция необходима при организации множественного отбора.

Если соответствующий элемент меню не активирован или при выходе из меню нажата клавиша Escape, меню-функции возвращают пустую строку или нуль (для числовых функций).

Отключение меню. Нижеперечисленные команды деактивируют/удаляют меню или его элементы.

- **DEACTIVATE POPUP** — деактивирует меню и удаляет его с экрана. Этот же результат может быть получен нажатием клавиши Escape или переходом к другому POPUP-меню. После деактивации меню управление передается команде, непосредственно следующей за **ACTIVATE POPUP**. Если обработка выбора из POPUP-меню осуществляется в специальной процедуре, последней командой, которая там выполняется, будет последняя команда процедуры, или команда **RETURN** (если есть), или команда **DEACTIVATE POPUP**. Если в меню

была нажата клавиша Escape, обращение к процедуре вообще не произойдет.

- **CLEAR POPUPS** — удаляет все деактивированные POPUP-меню с экрана и из памяти.
- **RELEASE POPUPS** [<список POPUP-меню> [EXTENDED]] — удаляет перечисленные деактивированные меню с экрана и из памяти. Если список опущен, удалятся все меню. Параметр EXTENDED вызовет удаление не только перечисленных POPUP-меню, но и всех подчиненных им вспомогательных меню.
- **RELEASE BAR** <вырN>/ALL OF <POPUP-меню> — команда удаляет перечисленные BAR-пункты из POPUP-меню. Можно удалить (ALL) и все BAR-пункты.
- **HIDE POPUP** <список POPUP-меню>/ALL [SAVE] — удаляет указанные или все (ALL) прямоугольные меню с экрана/окна, но не из памяти. Скрытые меню не равносильно деактивизации. Такое меню может быть восстановлено с помощью команд ACTIVATE/SHOW POPUP. Образ меню остается на экране, если указана опция SAVE. Очистить экран можно командой CLEAR.

Пр и м е р. Создадим POPUP-меню с именем MENU1 из четырех BAR-строк (пунктов):

ДОПОЛНЕНИЕ, КОРРЕКЦИЯ, УДАЛЕНИЕ, ОЧИСТКА БД, ВЫХОД

причем последний пункт отделен от остальных горизонтальной чертой.

Обработка пользовательского выбора из меню осуществляется процедурой PMENU1, в которую передаются значения имени меню, приглашения и номера пункта меню (POPUP(), PROMPT(), BAR()), выбранного пользователем. Если выбран пункт КОРРЕКЦИЯ, то процедура напечатает "MENU1 КОРРЕКЦИЯ 2 ". Вид самого меню изображен ниже справа. Первые три строки меню снабжены "горячими" клавишами — это первые буквы приглашений. Кроме того, пункты ДОПОЛНЕНИЕ и УДАЛЕНИЕ могут быть выбраны и другими "горячими" клавишами — Ctrl-N и Ctrl-T, обычно используемыми для этих целей. Пункт ОЧИСТКА БД ввиду его потенциальной опасности не имеет "горячих" клавиш и, кроме того, он снабжен предостерегающим сообщением. Чтобы привлечь внимание пользователя, и приглашение, и сообщение выкрашены в красный цвет, а последнее еще и мигает.

```
CLEAR
DEFINE POPUP menu1 FROM 5,35
DEFINE BAR 1 OF menu1 PROMPT '\<ДОПОЛНЕНИЕ' KEY Ctrl+N, '^N'
DEFINE BAR 2 OF menu1 PROMPT '\<КОРРЕКЦИЯ'
DEFINE BAR 3 OF menu1 PROMPT '\<УДАЛЕНИЕ' KEY Ctrl+T, '^T'
DEFINE BAR 4 OF menu1 PROMPT 'ОЧИСТКА БД';
MESSAGE 'Будьте осторожны' COLOR ,w+/r,...w+/r
DEFINE BAR 5 OF menu1 PROMPT '\-'

DEFINE BAR 6 OF menu1 PROMPT 'ВЫХОД';
ON SELECTION POPUP menu1 DO pmenu1 WITH POPUP(),PROMPT(),BAR()
ACTIVATE POPUP menu1
```

```
PROCEDURE pmenu1
PARAMETER mpopup,mprompt,mbar
? mpopup,mprompt,mbar
IF BAR()=6
DEACTIVATE POPUP
ENDIF
RETURN
```

ДОПОЛНЕНИЕ	^N
КОРРЕКЦИЯ	
УДАЛЕНИЕ	^T
ОЧИСТКА БД	
ВЫХОД	

Здесь сначала дается имя меню и указывается его место на экране (DEFINE POPUP), затем указываются номер и содержание каждого элемента меню (DEFINE BAR) с заданием "горячих" клавиш и, наконец, сообщается, что выбор из меню будет обрабатывать процедура PMENU1, в которую передаются три параметра (ON SELECTION...). После выбора любого пункта меню (кроме пункта ВЫХОД) осуществляется возврат в меню. При выборе последнего пункта (BAR()=6) командой DEACTIVATE POPUP работа с меню завершается и управление передается на команду, следующую за командой ACTIVATE POPUP. То же можно сделать и с помощью клавиши Escape, но процедура PMENU выполнена не будет. Команда/команды активации меню могут находиться в любом месте программы, где нужен доступ к меню, независимо от того, где и как оно было описано.

П р и м е р. Ниже приведена программа, в которой главное POPUP-меню KADRY может вызывать два вспомогательных меню BRIG и POISK (не показаны).

```
CLEAR
SET FULLPATH OFF
DEFINE POPUP kadry MARGIN                                && POPUP-меню KADRY
DEFINE BAR 1 OF kadry PROMPT 'Выход'
DEFINE BAR 2 OF kadry PROMPT 'Бригады';
SKIP FOR IIF(ADIR(x,'brig*.dbf')=0..t..f.)
DEFINE BAR 3 OF kadry PROMPT 'Поиск';
SKIP FOR !DBF()# 'D:\KADR.DBF'
ON SELECTION BAR 1 OF kadry DEACTIVATE POPUP
ON BAR 2 OF kadry ACTIVATE POPUP brig
ON BAR 3 OF kadry ACTIVATE POPUP poisk
DEFINE POPUP brig PROMPT FILES LIKE brig*.dbf          && POPUP-меню BRIG
ON SELECTION POPUP brig DO brig
DEFINE POPUP poisk                                       && POPUP-меню POISK
DEFINE BAR 1 OF poisk PROMPT 'Фамилия'
DEFINE BAR 2 OF poisk PROMPT 'Табель'
DEFINE BAR 3 OF poisk PROMPT 'Зарплата'
ON SELECTION POPUP poisk DO poisk
ACTIVATE POPUP kadry                                    && Активация меню
```

Пункты "Бригады" и "Поиск" снабжены средствами, ограничивающими к ним доступ, если это не имеет смысла. Так, первый пункт доступен только при наличии в текущей директории хотя бы одного бригадного файла, т.е. файла с именем, начинающимся со слова BRIG. Для этого в массив X заносятся все реквизиты таких файлов. Если размерность массива оказалась равной нулю, значит, подобных файлов нет. Пункт "Поиск" заблокирован, если в текущей рабочей области не открыта база KADR.DBF.

Вид меню с выбранными пунктами "Бригады" и "Поиск" приведен на рис.18.5 и 18.6.

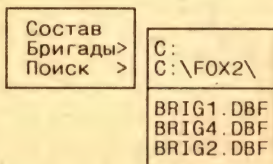


Рис.18.5

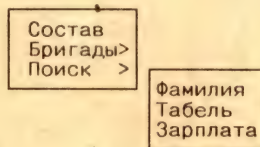


Рис.18.6

П р и м е р. Построим POPUP-меню с именем SOC, содержащее перечень четырех социальных учреждений. Меню должно допускать множественный выбор и отмечать отображенные пункты символом "*". Все отображенные объекты перед их пересылкой на обработку должны быть показаны справа от меню.

Клавишу Enter оставим в прежней роли, а действие окончания отбора закрепим за клавишей Space. Обработка этих нажатий выполняется в процедуре VIBOR. Если это клавиша Space (код 32), сканируются все пункты меню (FOR I=1 TO CNTBAR('SOC')), и если данный пункт помечен (IF MRKBAR('SOC',I)=T.), то он выводится на экран (?PRMBAR('SOC',I)) справа от меню. Программа приведена ниже, а вид экрана после окончания отбора — на рис.18.7.

```
SET TALK OFF
CLEAR
DEFINE POPUP soc FROM 5,0 MULTI MARGIN MARK '*' FOOTER;
    'Конец-Space' TITLE 'Социальные объекты';
    MESSAGE 'Отбор учреждений. Снятие пометки - Shift-Enter'
DEFINE BAR 1 OF soc PROMPT '\<Бассейн'
DEFINE BAR 2 OF soc PROMPT '\<Детский сад'
DEFINE BAR 3 OF soc PROMPT '\<Магазин'
DEFINE BAR 4 OF soc PROMPT '\<Поликлиника'
ON SELECTION POPUP soc DO vibor
ACTIVATE POPUP soc

PROCEDURE vibor  &&----- Процедура вывода
IF LASTKEY()=32
@ 5,20 CLEAR
FOR i=1 TO CNTBAR('soc')
    IF MRKBAR('soc',i)
        ? PRMBAR('soc',i) AT 20
    ENDIF
ENDFOR
DEACTIVATE POPUP  && Деактивация меню
ENDIF
RETURN
```

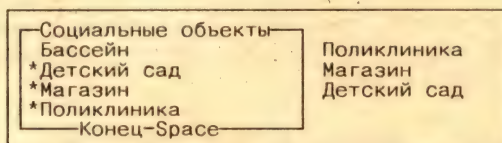


Рис.18.7

При работе с меню, содержащими имена файлов, может возникнуть одна опасность. Во-первых, из текущей (своей) директории пользователь может попасть в любую другую и "потеряться" там. Такая ситуация возникает при выборе пункта меню [...]. Кроме того, если он выберет самую верхнюю строку меню, содержащую имя текущего диска (например, C:), то ему (справа от основного меню) откроется перечень всех имеющихся дисков. Если пользователь выберет дисковод, в котором нет дискеты, например A:, то это вызовет системную ошибку ввода-вывода с соответствующим сообщением и возможным прерыванием программы. Чтобы этого избежать, можно заблокировать обработку таких пунктов меню.

Пр и м е р. Ниже приведена программа, в которой формируется POPUP-меню с именем FL. Меню содержит имена всех бригадных файлов (DBF-файлов, имена которых начинаются со слова BRIG). Само меню в положении, когда выбран пункт "C:", изображено справа. Чтобы избежать таких нежелательных выборов, используется не команда ON SELECTION POPUP, а процедура обработки выбора TTT, которую вызывает нажатие обеих клавиш Enter и Space (Spacebar). В указанной процедуре сначала анализируется вид самого правого элемента строки-приглашения (RIGHT(PROMPT(),1). Если это ":" или "]", значит, пользователь попытался сделать выбор пункта "C:" или "[...]". Такой выбор процедурой игнорируется. В противном случае выбор обрабатывается.


```

DEFINE POPUP fl PROMPT FILE LIKE brig*.dbf
ON KEY LABEL enter DO ttt
ON KEY LABEL spacebar DO ttt
ACTIVATE POPUP fl
ON KEY

PROCEDURE ttt
a=RIGHT(PROMPT(),1)
IF, a#'.AND.a#']'
    <обработка выбора>
ENDIF
RETURN

```

C:	>	A:
C:\FOX2\		B:
[...]		C:
BRIG1.DBF		D:
BRIG2.DBF		E:
BRIG3.DBF		
BRIG4.DBF		

Следующая команда перемещает активное меню в новое положение с абсолютными (Y,X) или относительными (вырN1,вырN2) координатами.

■ MOVE POPUP <POPUP-меню> TO <Y>,<X>/BY <вырN1>,<вырN2>

Команда может быть полезной для расположения меню таким образом, чтобы оно не закрывало от пользователя объект, с которым он работает.

Горизонтальное BAR-меню

Хотя BAR-меню по умолчанию — горизонтальное, его компоненты могут быть размещены в любых местах экрана. При создании меню используются команды:

Определение меню. Командой описания меню DEFINE MENU дается имя BAR-меню, а командой DEFINE PAD определяются его элементы (PAD-пункты). Команды ON SELECTION PAD и ON PAD определяют реакции меню на выбор клавишей Enter/Space.

Активация/деактивация меню. Меню, описанное указанными выше командами, может быть вызвано в любом месте программы командой ACTIVATE MENU. Затем оно должно быть отключено командой DEACTIVATE MENU или нажатием клавиши Escape. Если необходимость в меню отпала совсем, лучше его удалить из памяти командами CLEAR MENU или RELEASE MENU.

Рассмотрим эти команды.

Описание меню. Следующая команда дает имя BAR-меню, определяет его положение и вид

■ DEFINE MENU <имя меню> [BAR [AT LINE <вырN1>]]
 [IN [WINDOW] <окно>/IN SCREEN] [KEY <имя клавиши>]
 [MARK <вырC1>] [MESSAGE <вырC2>]
 [COLOR <список цветовых пар>/COLOR SCHEME <expN2>]

Параметр BAR указывает на то, что, если меню не уместится по ширине экрана/окна, оно будет доступно со скроллингом. Опция AT LINE <вырN> определяет строку, на которой появится меню (по умолчанию — нулевая строка). Меню с опцией BAR имеет свойства системного меню FoxPro. Оно хотя и допускает назначение KEY-клавиш для своих PAD-пунктов, но не предъявляет имена этих клавиш на экране. Выбор в таком меню завершается его деактивацией.

Определение PAD-пунктов меню. С помощью следующей команды BAR-меню может быть наполнено

- **DEFINE PAD** <имя PAD-элемента меню> **OF** <имя BAR-меню>
PROMPT <вырC1> [**AT** <Y,X>]
 [MESSAGE <вырC2>] [MARK <вырC3>]
 [KEY <имя клавиши> [, <вырC4>]]
 [SKIP [FOR <вырL>]]
 [COLOR <список цветовых пар>/COLOR SCHEME <вырN>]

Команда дает имя PAD-элементу меню, указывает на его принадлежность какому-то BAR-меню, определяет его отображение (вырC1) и место (Y,X) на экране. Если параметр AT опущен, все PAD-элементы меню будут располагаться в нулевой строке экрана слева направо.

Определение реакций меню. Следующая команда при любом выборе из BAR-меню реализует указанную <команду>:

- **ON SELECTION MENU** <BAR-меню> [<команда>]

Если параметр <команда> опущен, назначение отменяется.
 Команда

- **ON SELECTION PAD** <PAD-пункт> **OF** <BAR-меню> [<команда>]

позволяет делать индивидуальные назначения команд на каждый <PAD-пункт> из <BAR-меню>. Обычно это <команды> DO вызова процедуры, где может быть установлено, какой именно был выбран PAD-элемент. Выбор указанного пункта меню требует нажатия клавиши Enter/Space.

Команда **ON PAD** назначает на <PAD-пункт меню> из главного горизонтального <BAR-меню> дополнительное меню следующего уровня <POPUP-меню>/<BAR-меню1>:

- **ON PAD** <PAD-пункт> **OF** <BAR-меню>
 [ACTIVATE POPUP <POPUP-меню>
 /ACTIVATE MENU <BAR-меню1>]

Теперь, если вы выбираете пункт из горизонтального BAR-меню, будет автоматически отображаться и соответствующее вспомогательное POPUP/BAR-меню. Такое меню должно быть определено командой **DEFINE POPUP/MENU** до того, как оно будет использовано в команде **ON PAD**. Меню второго уровня может быть и другим горизонтальным меню <BAR-меню1>, формирующим новые уровни системы сложного иерархического меню.

Вместо команды **ON PAD** можно использовать и команду **ON SELECTION PAD**, но при этом вспомогательное POPUP-меню будет появляться не автоматически, а при нажатии клавиши Enter/Space.

Для удаления связи всплывающего меню с <PAD-пунктом> меню можно ввести команду **ON PAD** без параметра **ACTIVATE POPUP**.

Функции выбора из меню. Для передачи в процедуры параметров могут использоваться функции:

- **MENU()** — Возвращает имя активного BAR-меню.
- **PAD()** — Возвращает имя PAD-пункта, выбранного в BAR-меню.
- **PROMPT()** — Возвращает строку-приглашение, содержащуюся в PAD-элементе, выбранном в BAR-меню.
- **CNTPAD**(<BAR-меню>) — Возвращает количество PAD-пунктов в BAR-меню.

- MRKPAD(<BAR-меню>,<PAD-пункт>) — Возвращает значение .T., если <PAD-элемент> в <BAR-меню> помечен, и .F. — если нет.
Отключение меню. Следующие команды деактивируют/удаляют меню:
- DEACTIVATE MENU — удаление активного BAR-меню с экрана (но не из памяти), переход на команду, следующую за ACTIVATE MENU.
- RELEASE MENUS [<список BAR-меню> [EXTENDED]] — удаление всех или перечисленных неактивных BAR-меню с экрана и из памяти, включая и все подчиненные меню (если указана опция EXTENDED).
- CLEAR MENUS — удаление неактивных меню с экрана и из памяти.
- RELEASE PAD <PAD-пункт>/ALL OF <имя BAR-меню> — удаляет указанный PAD-пункт BAR-меню из памяти и с экрана/окна. Можно удалить (ALL) и все PAD-пункты меню (кроме системного меню). Включение опции EXTENDED удаляет также и все подчиненные меню следующего уровня.
- HIDE MENU <список BAR-меню>/ALL [SAVE] — удаляет указанные или все (ALL) горизонтальные меню с экрана/окна, но не из памяти. Скрытие меню не равнозначно деактивизации. Такое меню может быть восстановлено с помощью команд ACTIVATE/SHOW MENU. Образ меню остается на экране, если указана опция SAVE. Очистить экран можно командой CLEAR.

Любое меню можно покинуть, нажав клавишу Escape или используя команду DEACTIVATE. В этих случаях управление передается на команду, непосредственно следующую за командой ACTIVATE. Команда DEACTIVATE прекращает процедуру, в которой она находится, и никакие следующие команды внутри процедуры выполняться не будут. Естественное завершение процедуры (по команде RETURN или при достижении конца процедуры) возвращает нас к команде ACTIVATE.

Пример. Построим горизонтальное меню с именем MAINMENU и содержимым

РЕДАКТИРОВАНИЕ ПОИСК КОНЕЦ

При выборе с помощью клавиши Enter любого из пунктов меню вызывается процедура PPP, в которую передаются параметры MENU() и PAD(), где они и печатаются. Если сделан выбор пункта ПОИСК, будет выведена строка "MAINMENU POISC".

```
CLEAR
DEFINE MENU mainmenu
DEFINE PAD redak OF mainmenu PROMPT ' \<РЕДАКТИРОВАНИЕ' AT 1,10
DEFINE PAD poisk OF mainmenu PROMPT ' \<ПОИСК ' AT 1,32
DEFINE PAD konec OF mainmenu PROMPT ' \<КОНЕЦ ' AT 1,50
ON SELECTION PAD redak OF mainmenu DO ppp WITH MENU(),PAD()
ON SELECTION PAD poisk OF mainmenu DO ppp WITH MENU(),PAD()
ON SELECTION PAD konec OF mainmenu DO ppp WITH MENU(),PAD()
ACTIVATE MENU mainmenu
```

```
PROCEDURE ppp
PARAMETER mmenu, mpad
? mmenu,mpad
RETURN
```

Вызванная процедура после своего завершения возвращает управление в вызывающую программу на строку ACTIVATE MAINMENU до нажатия клавиши Escape.

Замечание. Поскольку здесь во всех случаях идет обращение к процедуре PPP, в данном примере имеет также смысл использовать только одну команду

ON SELECTION MENU mainmenu DO ppp WITH MENU(), PAD()
вместо трех команд ON SELECTION PAD.

Двухуровневое PULLDOWN-меню

Двухуровневое меню создается с помощью уже указанных команд в следующей последовательности.

Командами DEFINE MENU и DEFINE PAD описывается главное горизонтально меню.

Командой ON SELECTIN PAD обычно описываются реакции тех пунктов меню, которые вызывают процедуры, т.е. те выходы меню, которые заканчиваются на первом (горизонтальном) уровне. При этом для их выбора требуется нажатие клавиши Enter.

Командами ON PAD с опцией ACTIVATE POPUP устанавливается связь горизонтального меню с вертикальными POPUP-меню, т.е. выход на второй уровень меню.

Командами DEFINE POPUP FIELD/FILES/STRUCTURE или DEFINE POPUP с командами DEFINE BAR описываются сами POPUP-меню, а их реакции — командой ON SELECTION POPUP.

Активация всей конструкции меню осуществляется командой ACTIVATE MENU, деактивация — командой DEACTIVATE MENU.

Анализ выбора пользователя может выполняться с помощью функций BAR(), POPUP(), PROMPT(), MENU(), PAD(), ...

Организацию взаимодействия компонентов меню иллюстрирует рис.18.8.

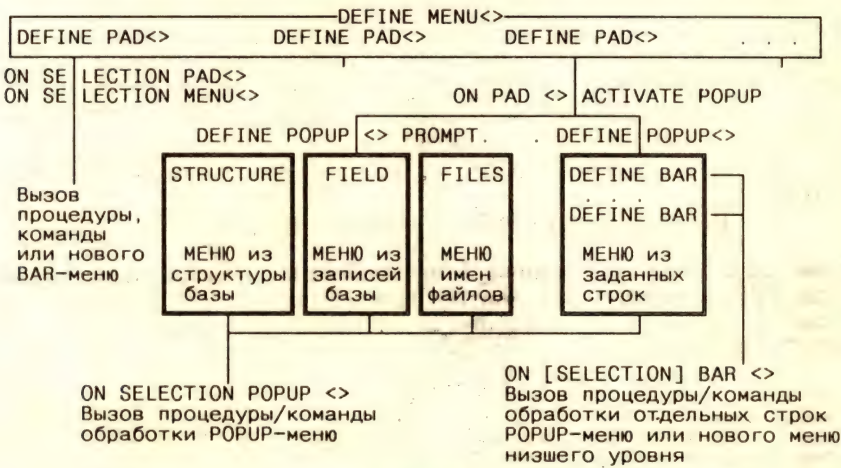


Рис.18.8

Пр и м е р. Построим сложное четырехуровневое меню, изображенное на рис.18.9 (уровни меню указаны цифрами).

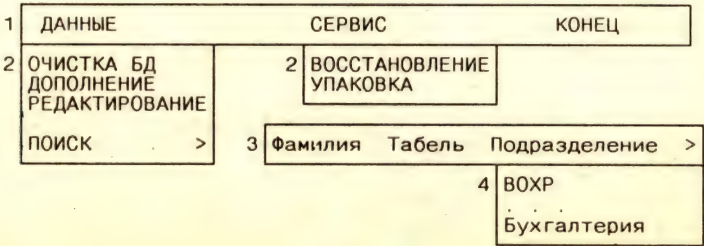


Рис.18.9

Здесь главное горизонтальное BAR-меню (MENU0) содержит пункты: ДАННЫЕ, СЕРВИС, КОНЕЦ.

Пункту ДАННЫЕ отвечает вспомогательное POPUP-меню MDAN второго уровня (ОЧИСТКА БД, ДОПОЛНЕНИЕ, РЕДАКТИРОВАНИЕ, ПОИСК). ОЧИСТКА реализуется одной командой ZAP, ДОПОЛНЕНИЕ и РЕДАКТИРОВАНИЕ — процедурой PDAN. При попадании в пункт ПОИСК раскрывается BAR-меню третьего уровня MPSK (пункты: Фамилия, Табель, Подразделение). В пункте Подразделение можно вызвать POPUP-меню MPODR четвертого уровня, которое предъявляет все подразделения базы KADR.DBF. Для того чтобы пользователь знал о существовании меню четвертого уровня, пункт Подразделения BAR-меню MPSK снабжен справа значком ">", который обычно указывает на возможность вызова к данному пункту меню какого-то меню следующего уровня. В POPUP-меню это делается без нашего участия, но в BAR-меню мы должны позаботиться об этом сами.

Выбранное подразделение ищется процедурой PPODR. Поиск Фамилии и Табельного номера обрабатывается в процедуре PPSK.

Пункт СЕРВИС главного меню вызывает POPUP-меню второго уровня MSERV, где могут быть выполнены ВОССТАНОВЛЕНИЕ (переиндексация) поврежденных индексов и УПАКОВКА базы KADR.DBF. Эти действия реализованы просто командами REINDEX и PACK.

Пункт КОНЕЦ завершает программу командой CANCEL.

Ниже приведена программа, реализующая описанную систему меню. Вызываемые процедуры не показаны, но, конечно, должны существовать. Здесь они только вызываются, причем в некоторые из них функциями PROMPT(), PADO(), MENU(), RECNO(), BAR() передаются и параметры.

```
SET TALK OFF
CLEAR
USE kadr INDEX kadrfam, kadrta IN a
USE podr IN b
DEFINE MENU menu0 && Определение главного BAR-меню MENU0
DEFINE PAD dann OF menu0 PROMPT '\<ДАННЫЕ' AT 1,10
DEFINE PAD serv OF menu0 PROMPT '\<СЕРВИС' AT 1,32
DEFINE PAD konec OF menu0 PROMPT '\<КОНЕЦ' AT 1,50
* Вызов вспомогательного POPUP-меню MSERV к PAD-пункту СЕРВИС
ON PAD serv OF menu0 ACTIVATE POPUP mserv
* Вызов вспомогательного POPUP-меню MDAN к PAD-пункту ДАННЫЕ
ON PAD dann OF menu0 ACTIVATE POPUP mdan
ON SELECTION PAD konec OF menu0 CANCEL && Конец

DEFINE POPUP mserv && Определение POPUP-меню MSERV
DEFINE BAR 1 OF mserv PROMPT '\<ВОССТАНОВЛЕНИЕ'
DEFINE BAR 2 OF mserv PROMPT '\<УПАКОВКА'
ON SELECTION BAR 1 OF mserv REINDEX && Переиндексация
ON SELECTION BAR 2 OF mserv PACK && Упаковка

DEFINE POPUP mdan && Определение POPUP-меню MDAN
DEFINE BAR 1 OF mdan PROMPT '\<ОЧИСТКА БД'
DEFINE BAR 2 OF mdan PROMPT '\<ДОПОЛНЕНИЕ'
DEFINE BAR 3 OF mdan PROMPT '\<РЕДАКТИРОВАНИЕ'
DEFINE BAR 4 OF mdan PROMPT '\<ПОИСК'
ON BAR 1 OF mdan ZAP && Очистка базы (к BAR-пункту 1)
* Вызов вспомогательного BAR-меню MPSK (к BAR-пункту 4)
ON BAR 4 OF mdan ACTIVATE MENU mpsk

DEFINE MENU mpsk && Определение BAR-меню MPSK
DEFINE PAD fam OF mpsk PROMPT '\<Фамилия' AT 7,28
DEFINE PAD tab OF mpsk PROMPT '\<Табель'
DEFINE PAD pod OF mpsk PROMPT '\<Подразделение >'
* Вызов вспомогательного POPUP-меню PODR к пункту Подразделение
ON SELECTION PAD pod OF mpsk ACTIVATE POPUP mpodr
* Вызов процедуры PPSK для обработки остальных PAD-пунктов меню
ON SELECTION MENU mpsk DO ppsk
```

* Определение POPUP-меню MPODR для предъявления подразделений


```

DEFINE POPUP mpoдр FROM 8,47 PROMPT FIELD b.podr
* Вызов процедуры PDAN для обработки остальных пунктов BAR-меню
ON SELECTION POPUP mpoдр DO ppoдр;
  WITH PROMPT(),PAD(),MENU(),RECNO()

```

ACTIVATE MENU menu0 && Активация всей системы меню

На рис.18.10 приведена структура описанной системы меню с указанием имен меню и закрепленных за его пунктами команд (в скобках).

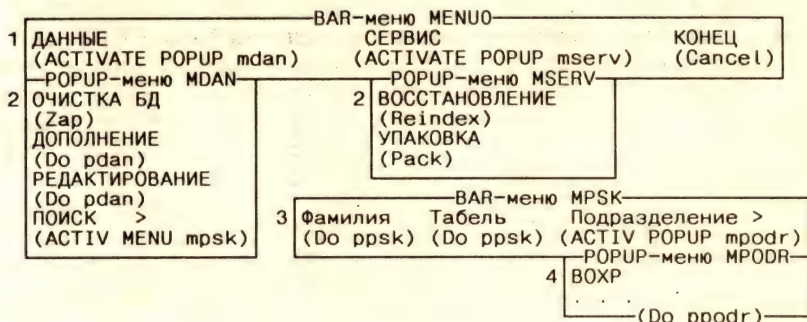


Рис.18.10

При написании сложных меню-программ, вероятно, следует начать именно с этого.

Управление доступом к меню. Кроме управления доступом к некоторым пунктам меню с помощью опций SKIP [FOR <вырL>] команды DEFINE в FoxPro имеются следующие специальные команды разрешения доступа:

- SET SKIP OF MENU <BAR-меню> <вырL>
- SET SKIP OF PAD <PAD-пункт> OF <BAR-меню> <вырL>
- SET SKIP OF POPUP <POPUP-меню> <вырL>
- SET SKIP OF BAR <вырN> OF <POPUP-меню> <вырL>

которые позволяют в зависимости от условия (<вырL>) включать/выключать <BAR-меню>, <PAD-пункт>, <POPUP-меню> и отдельные (имеющие номер <вырN>) BAR-строки POPUP-меню соответственно. Эти элементы будут показаны на экране приглушенным цветом. Они не могут быть выбраны, если <вырL>=.T.. Отмена блокировки выполняется той же командой, но с аргументом .F.. Перечисленные команды можно вносить в состав любых процедур, в том числе и вызываемых данным меню, что позволит динамически изменять его облик в зависимости от обстоятельств.

В заключение подведем некоторые итоги по применению рассмотренной технологии меню.

○ Здесь нет необходимости в постоянном возобновлении команды активации меню (как, например, команды READ MENU BAR TO ... в Fox-меню), а следовательно, и в цикле DO WHILE .t. ... ENDDO, поскольку в dBASE-меню меню остаются активными до их явной деактивации.

○ Можно легко построить не только двухуровневое меню, но и иерархическое меню практически любой сложности, используя лишь средства описания меню и совершенно не прибегая к программированию связей между ними.

○ Имеется возможность сделать меню чувствительным к текущей обстановке и даже поддерживать "обратные связи" вызываемых процедур с меню. Средства блокирования/разблокирования целых меню и их элементов предоставляют опция SKIP в командах описания меню и команды вида SET

SKIP OF. Кроме того, можно управлять и видом самого меню. Так, в PROMPT-строки можно вставлять функции, в частности функцию IIF(), которая в зависимости от некоторых условий может устанавливать то или иное приглашение. Например, в зависимости от переменной X функция IIF() может предъявлять строку "Вход" или "Выход" (см. фрагмент программы ниже). Однако это условие опрашивается только при первоначальной активации меню. В дальнейшем изменение значения X не повлечет изменения меню. Если же вы хотите управлять обликом меню из вызываемых им процедур, туда следует поместить команду переопределения нужных строк меню (процедура VH).

```
x=.t.
DEFINE POPUP ...

DEFINE BAR ... OF ... PROMPT IIF(x, 'Вход', 'Выход')
ON SELECTION POPUP ... DO vh
ACTIVATE POPUP ...

PROCEDURE vh

x=!x
DEFINE BAR ... OF .... PROMPT IIF(x, 'Вход', 'Выход')
RETURN
```

Выбор рассматриваемого пункта меню каждый раз будет немедленно изменять слово "Вход" на "Выход" и наоборот.

О Важным является вопрос о включении средств обработки меню в тело программы. Это можно сделать различным образом, так что меню может вызываться и обрабатываться как внутри одной и той же процедуры, так и/или во многих независимых процедурах/программах.

Функции основной программы можно "рассыпать" по процедурам. Для программиста это удобно, поскольку позволяет легче контролировать процесс создания прикладной системы. Вместе с тем, если процедур слишком много, они могут замедлить работу готовой системы и затруднить ориентирование в ней разработчика. При желании некоторые или все процедуры можно слить в одну. В этом случае для того, чтобы была возможность различить в структуре DO CASE, откуда пришел вызов, в нее, возможно, придется передать не только имя или номер пункта, выбранного во вспомогательном меню (например, POPUP-меню), но и имя пункта старшего меню.

Рассмотрим вопрос о процедурах на примере POPUP-меню. Пусть командой DEFINE POPUP <POPUP-меню> определено некоторое прямоугольное меню с именем <POPUP-меню>. Следующие команды ON SELECTION POPUP реализуют разные варианты обработки выбора из меню, вызванного командой ACTIVATE POPUP <POPUP-меню>.

1. ON SELECTION POPUP <POPUP-меню>;
DEACTIVATE POPUP <POPUP-меню>

Команда ON SELECTION в такой форме имеет смысл если предполагается, что обработка выбора будет выполняться не в некоторой отдельной процедуре, а внутри той же самой процедуры, где было вызвано меню (т.е. встретилась команда ACTIVATE POPUP). После активации меню и осуществления выбора меню деактивируется без вызова какой-либо процедуры и будет выполняться команда, стоящая непосредственно за командой ACTIVATE POPUP <POPUP-меню>.

2. ON SELECTION POPUP ALL DO <имя процедуры>

Все активные <POPUP-меню> обрабатываются в одной <процедуре>.

3. ON SELECTION POPUP <POPUP-меню> DO <имя процедуры>

Все пункты указанного <POPUP-меню> обрабатываются в поименованной

<процедуре>.

```
4. ON SELECTION POPUP <POPUP-меню> DO <имя процедуры0>  
   ON SELECTION BAR <вырN1> OF <POPUP-меню> DO <имя процедуры1>  
   ON SELECTION BAR <вырN2> OF <POPUP-меню> DO <имя процедуры2>  
   ON BAR <вырN3> OF <POPUP-меню> ACTIVATE POPUP <POPUP-меню3>  
   ON SELECTION BAR <вырN4> OF <POPUP-меню>
```

Выбор BAR-строк меню номер <вырN1> и <вырN2> обрабатывается в <процедурах 1 и 2>. Выбор строки номер <вырN3> активирует другое <POPUP-меню3>. Выбор остальных строк (с номерами <вырN4> <вырN5> и др.) будет анализироваться в указанной <процедуре0>, если соответствующие ON-команды не содержат ссылок на какую-то другую процедуру или команду.

Перечисленные возможности могут быть реализованы также для BAR-меню и систем иерархических меню.

5. Активация любого меню возможна также нажатием некоторой назначенной клавиши, если при его описании использовалась опция KEY. Это ускоряет доступ к меню и делает его возможным из любого места программы.

Как видим, FoxPro предоставляет исключительно гибкий и мощный инструмент управления программой.

Вообще выбор вида меню далее определяет всю структуру прикладной системы. При создании систем обработки данных, безусловно, удобнее использовать dBASE-меню. Вместе с тем в отдельных случаях может быть полезным и Fox-меню.

18.2. Клавишное меню

В отличие от светового меню клавишное меню не подразумевает никаких движущихся элементов. Обычно это просто строка, где содержится перечень возможных действий и вызывающих их клавиш.

Меню, построенные таким образом, как правило, используют для реализации простых, но часто выполняемых операций по обработке данных. При этом можно выбрать некоторые наиболее удобные или привычные клавиши на клавиатуре. Клавишное меню в особенности полезно для вызова определенных действий без того, чтобы покидать текущий экран, например экран редактирования. Ниже рассмотрены команды, необходимые для построения такого меню.

Команды связи с клавиатурой и прерываниями от ошибок

Рассматриваемые команды позволяют установить связь программы с прерываниями от нажатия клавиш или возникновения ошибок (прерывание ERROR) с переходом к выполнению указанных в них <команд>.

- ON ERROR [<команда>] — переход к выполнению <команды> при возникновении любой ошибки.
- ON READERROR [<команда>] — то же, но при ошибке ввода. <Команда> выполняется в случае, если введены неправильные данные или данные, выходящие за диапазон, заданный в RANGE, или не отвечающие условию VALID.
- ON ESCAPE [<команда>] — переход к выполнению <команды> при нажатии клавиши Escape. Если SET ESCAPE OFF, команда не работает.
- ON KEY [<команда>] — то же, при нажатии любой клавиши.
- ON KEY=<вырN> [<команда>] — то же, но при нажатии клавиши с кодом, соответствующим <вырN>.
- ON KEY [LABEL <имя клавиши>] [<команда>] — то же, но при нажатии

клавиши с указанным <именем>.

Каждая ON-команда имеет силу до тех пор, пока не появится другая команда ON такого же типа с новым значением параметра <команда>. Полная отмена ON-команды осуществляется командой такого же типа, но без параметров.

Хотя <команда> может быть командой любого типа, обычно это команда DO вызова процедуры/программы, после завершения которой осуществляется возврат на команду, следующую за той, от которой произошел вызов. Исключение составляют процедуры, завершающиеся командой RETRY. В этом случае произойдет возврат на команду, где было вызвано прерывание.

Команды, перехватывающие управление при ошибке, можно использовать для предотвращения соответствующих системных сообщений FoxPro, устранения причин ошибки или организации "культурного" выхода из программы. Последнее особенно важно, так как пользователь, неожиданно увидев на экране обычно непонятное ему системное сообщение на английском языке пугается, а иногда и пытается перезагрузить компьютер, что может повлечь потерю данных.

Самая очевидная область применения команд обработки клавиш — это создание клавишных меню, совмещенных с экранами редактирования, которые сформированы с помощью команд BROWSE, EDIT, CHANGE, READ.

Команда ON KEY реагирует на нажатие любой клавиши, однако не в режиме редактирования данных.

В командах ON KEY= и ON KEY LABEL можно указать практически любые клавиши или комбинации клавиш.

В программе может одновременно быть активной только одна команда вида ON KEY=, т.е. ею задействуется только одна клавиша.

Напротив, любое число команд ON KEY LABEL может активировать любое число клавиш одновременно. Кроме того, в этой команде клавиши активируются не по кодам (<вырN>), которые невозможно запомнить, а по своим именам. Эти имена легко указать, так как они практически совпадают с надписями на клавишах (рис.18.11).

Имя клавиши может быть задано как строчными, так и прописными буквами. Например, команда

```
ON KEY LABEL ctrl+f DO prog
```

при нажатии клавиш CTRL-F вызовет процедуру PROG или программу PROG.PRG.

Допускается в качестве активных клавиш использовать любые, а не только управляющие. Именем такой клавиши является символ, нанесенный на нее (A — Z, 0 — 9 и т.д.).

Таким образом, команда ON KEY LABEL является самым удобным средством организации клавишных меню. Однако имеется одна особенность, которую следует учитывать при работе. Клавиши, определенные в команде ON KEY LABEL, остаются (до отмены) активными и доступными всегда. Нажатие такой клавиши влечет вызов назначенной ей процедуры в любом месте исполняемой программы, в том числе и не в состоянии ожидания. При этом могут быть ошибочно прерваны какие-то процессы и вызвана ненужная в данном случае процедура. Например, возможно даже рекурсивное обращение клавишной процедуры к самой себе, если активная клавиша была нажата дважды. Чтобы этого избежать, необходимо своевременно отключать активные клавиши (сразу все — командой ON KEY или некоторые — командами ON KEY LABEL <клавиша> без параметра <команда>) в программе и включать их снова, когда они понадобятся.

Для этих целей удобно также использовать следующие две команды

удаления/восстановления клавишных назначений

- PUSH KEY [CLEAR]
- POP KEY [ALL]

Клавиши	Имена клавиш
Стрелка влево	LEFTARROW
Стрелка вправо	RIGHTARROW
Стрелка вверх	UPARROW
Стрелка вниз	DNARROW
Home	HOME
End	END
PgUp	PGUP
PgDn	PGDN
Del	DEL
Backspace	BACKSPACE
Ins	INS
Tab	TAB
Shift Tab	BACKTAB
Enter	ENTER
с F1 по F10	F1 F10
с Ctrl-F1 по Ctrl-F10	Ctrl+F1 - Ctrl+F10
с Shift-F1 по Shift-F9	Shift+F1 - Shift+F9
с Alt-F1 по Alt F10	Alt+F1 - Alt+F10
с Alt-0 по Alt-9	Alt+0 - Alt+9
с Alt-A по Alt-Z	Alt+A - Alt+Z
Любая кнопка мыши	MOUSE
Левая кнопка мыши	LEFTMOUSE
Правая кнопка мыши	RIGHTMOUSE
Ctrl-влево	CTRL+LEFTARROW
Ctrl-вправо	CTRL+RIGHTARROW
Ctrl-Home	CTRL+HOME
Ctrl-End	CTRL+END
Ctrl-PgUp	CTRL+PGUP
Ctrl-PgDn	CTRL+PGDN
с Ctrl-A по Ctrl-Z	CTRL+A - CTRL+Z
{	LBACE
}	RBACE
Пробел	SPACEBAR

Рис.18.11

Команда PUSH KEY сохраняет все текущие клавишные назначения, сделанные командами ON KEY LABEL в стеке памяти компьютера, одновременно отменяя их действие, если указана опция CLEAR. Несколькими последовательными командами PUSH KEY в стек можно поместить несколько отдельных групп клавишных назначений. Оттуда при необходимости они могут быть извлечены в обратном порядке командой POP KEY и снова сделаны активными. Если задана опция ALL, стек очищается и отменяются все текущие назначения.

Текущие клавишные назначения могут быть просмотрены командами DISPLAY/LIST STATUS или с помощью функции ON().

Пр и м е р. Пусть производится назначение на клавиши F3 и F5 сначала команд CLEAR и DIR, а затем на F5 команды DO KADR. Эти назначения последовательно сохраняются в стеке командами PUSH KEY и извлекаются из него командами POP KEY. Активные назначения предъявляются функцией ON(). Результаты ее возможного применения выведены в два столбца после знаков "&&". В левом столбце назначение клавиши F3, в правом — F5.

```
POP KEY ALL
ON KEY LABEL F3 CLEAR
ON KEY LABEL F5 DIR
? ON('KEY', 'F3'), ON('KEY', 'F5') && CLEAR, DIR
PUSH KEY CLEAR
```

&& Сброс клавиш, очистка стека


```

ON KEY LABEL F5 DO kadr      &&      , DO kadr
PUSH KEY CLEAR               &&      ,
POP KEY                      &&      , DO kadr
POP KEY                      && CLEAR, DIR
POP KEY                      &&

```

В дальнейшем будет показано много примеров использования команд ON KEY LABEL в программах.

18.3. Создание меню с помощью функций

Самые простые ждущие клавишные меню можно строить, используя функции INKEY(), LASTKEY(), READKEY(). При этом предварительно должно быть создано состояние ожидания.

Функция INKEY() такое состояние создает сама. Приведем пример меню, использующего функциональные клавиши:

```

?'F2 - удаление F3 - возврат F4 - выход'
x=INKEY(0)
DO CASE
  CASE LASTKEY()=-3      && Нажата клавиша F2
  CASE LASTKEY()=-4      && Нажата клавиша F3
ENDCASE

```

Здесь сначала демонстрируется текст меню, а затем код нажатой клавиши запоминается в переменной X, которая дальше анализируется.

Средством создания паузы может быть и команда WAIT (лучше с опцией WINDOW), например:

```

WAIT 'Esc - отказ, ПРОДОЛЖЕНИЕ - любая клавиша' WINDOW
IF LASTKEY()=27
  RETURN
ENDIF

```

Здесь программа останавливается и ожидает управляющих действий пользователя. При нажатии клавиши Escape происходит выход из процедуры (RETURN).

Анализ нажатий в окне редактирования. Функция READKEY() похожа на LASTKEY() и позволяет сделать выбор альтернатив дальнейшей обработки данных непосредственно в окне редактирования READ, поскольку она различает не только, какие клавиши были нажаты для выхода из экрана ввода, но и были ли в нем изменены данные.

Положим, что в зависимости от содержимого текущей записи, которая предъявлена на редактирование, мы должны сделать один из следующих выборов:

- поставить запись на удаление;
 - переместиться на следующую запись;
 - прекратить просмотр и редактирование записей в случае, если данные были изменены;
 - продолжить естественный ход выполнения программы.
- Схема такой программы приведена ниже (экран ввода не показан).

```
DO WHILE .T.
```

Экран ввода-редактирования

```

@ 20,1 SAY 'Ctrl-Home - удаление, Ctrl-PgUp - дальше, Ctrl-End - конец'
READ
c=READKEY()      && c=LASTKEY()
DO CASE

```



```

CASE c=33.OR.c=289      && c=29
  DELETE
CASE c=34.OR.c=290      && c=31
  SKIP
CASE c=270              && c=23.AND.UPDATE()
  EXIT
ENDCASE

```

```

ENDDO

```

В нижней части экрана ввода изображено меню, в котором указаны задействованные в нем клавиши. После выхода из экрана обычным образом или нажатием одной из перечисленных комбинаций клавиш анализируется функция READKEY(). Если она возвращает значение 33 или 289, значит, были нажаты клавиши Ctrl-Home, если 34 или 290 — значит, Ctrl-PgUp, если 270 — значит, Ctrl-End и данные были изменены. В зависимости от этого далее выполняются различные действия по обработке данных. Однако свободных клавиш выхода из команд редактирования, которые можно использовать таким образом, очень немного.

Вообще возможности функции READKEY() значительно шире. С ее помощью можно выяснить и другие обстоятельства завершения редактирования, что может быть очень полезным при организации интерфейса, одновременно использующего несколько команд READ.

Аналогичным образом может быть применена функция LASTKEY() (альтернативные команды приведены справа после знаков "&&"), а факт обновления данных может быть зафиксирован функцией UPDATED().

Очевидным неудобством программирования такого интерфейса является необходимость в цикле DO WHILE...ENDDO для регенерации содержимого окна редактирования после того как, содержимое текущих полей было изменено, например был перемещен указатель записей. В FoxPro теперь благодаря новым опциям команды READ и новым командам SHOW GETS/GET/OBJECT реализована возможность обновления данных без выхода из READ.

18.4. Использование функциональных клавиш

Если вы нажмете любую функциональную клавишу компьютера, то увидите, что сначала покажется в командном окне, а затем и исполнится какая-то команда. Например, нажатие F3 повлечет выполнение команды LIST. Эту возможность можно использовать и для создания клавишных меню с помощью команды

■ SET FUNCTION <вырN>/<имя клавиши> TO [<вырC>]

Команда закрепляет за функциональной клавишей, указанной номером <вырN> или <именем клавиши>, определенное выражение символьного типа. Включение в него знака ";" будет соответствовать нажатию клавиши Enter. Допускается использование комбинации функциональных клавиш с клавишами Ctrl и Shift.

Например, команда

```

SET FUNCTION Ctrl+F4 TO 'do kadr;'

```

повлечет выполнение программы KADR.PRG при нажатии клавиш Ctrl-F4. Здесь может быть задано сразу несколько команд FoxPro, перечисленных через знак ";". Такое закрепление последовательности команд (макрокоманд) может быть полезно, но только при работе в системном интерфейсе через командное окно.

Хотя выражение после слова **TO** должно быть символьного типа, если оно имеет числовой вид, оно правильно принимается командами ввода **@...GET**. Например, в следующем фрагменте программы нажатие **F7** влечет занесение в числовую переменную **R** числа **8145.86**. Знак **”;** для числовых переменных не распознается. Такое закрепление может быть удобно для ввода часто повторяющихся данных.

```
SET FUNCTION F7 TO '8145.86'
@ 5,6 GET r DEFAULT 0
READ
```

По умолчанию в **FoxPro** за **F**-клавишами закреплены определенные команды (**Help**, **Set**, **List** и т.д.). Чтобы их подавить, нужно воспользоваться командой

■ CLEAR MACROS

Этой же командой отменяются и все программные назначения **F**-клавиш. Индивидуальная отмена назначений может быть выполнена командой **SET FUNCTION <клавиша> TO** без указания параметра.

18.5. Дополнительные вопросы

Множественный выбор в базе данных. Поскольку **POPUP**-меню с опцией **PROMPT FIELD** не обеспечивает множественного отбора, приходится прибегать к программированию этой функции. Здесь возможно несколько подходов. Рассмотрим их на примере реализации множественного выбора в базе **KADR.DBF**.

Для запоминания факта выбора необходимо ввести в базу дополнительное логическое поле **V**. Его содержимым будет **.T.**, если данный человек отобран, и **.F.** в противном случае. Опция **PROMPT FIELD fam+IIF(v,'+', ' ')+pr()** в команде описания меню указывает, что кроме фамилии в меню будет показано поле **V**, но не непосредственно, а с помощью знака **+**, если запись отобрана, и пробел — если нет. Последнее более удобно для отечественного пользователя, чем вывод **.T./F.**

```
USE kadr
DEFINE POPUP kadr PROMPT FIELD fam+IIF(v,'+', ' ')+pr()
ON SELECTION POPUP kadr REPLACE v WITH !v
ACTIVATE POPUP kadr
```

```
FUNCTION pr
@ 20,0
@ 20,0 SAY podr
RETURN ''
```

В команде **ON SELECTION POPUP** предусмотрена замена значения поля **V** на противоположное. Замена **.F.** на **.T.** нужна для пометки отобранного человека, а **.T.** на **.F.** — на случай, если выбор сделан по ошибке (отмена выбора). Далее отобранные поля могут быть легко извлечены из базы данных.

Используя возможность указания в опции **PROMPT FIELD** команды описания меню любых выражений, можно реализовать дополнительные функции, отслеживающие текущее положение курсора в меню еще до того, как сделан какой-либо выбор, например вывод дополнительных сообщений, более полно раскрывающих текущий пункт меню.

В данном случае с помощью функции **PR()** в нижней части экрана выводится мемо-поле **PER** (перемещения).

При множественном выборе необходимо учитывать два момента, влияющих

на скорость работы готовых приложений, — извлечение отобранных записей и последующую очистку поля с признаком отбора. Для этого, возможно, придется дважды прогонять всю базу данных, что при ее заметных размерах потребует значительных потерь времени. В особенности, конечно, желательно избавиться от раздражающей пользователя задержки перед каждой активацией меню, вызванной необходимостью сброса поля признака выбора (в нашем случае для поля V — это REPLACE ALL v WITH .f.).

Здесь можно поступить различным образом. Прежде всего следует попытаться совместить некоторые из указанных процессов, например отбор и обработку записей или обработку и снятие пометки. Однако технология работы прикладной системы может этого и не позволить. Ниже предлагаются некоторые возможные решения, освобождающие нас от необходимости очистки поля V.

Можно прибегнуть к хранению в поле V не значения .T./F., а даты и времени. Если эту информацию сохранять в виде ГГГГ.ММ.ДД.ЧЧ.ММ.СС, то для поля V (уже символьного типа) потребуется четырнадцать позиций. Для выбранных записей поле V получит значение даты и времени активации меню. Для всех остальных оно будет иметь некоторое меньшее значение (даты предыдущих отборов).

Программа приведена ниже. Здесь в процедуре DAT в переменной C формируется и хранится *дата+время*. Данная процедура должна обязательно выполняться каждый раз перед активацией меню.

```
USE kadr
C=' '
DEFINE POPUP kadr PROMPT FIELD fam+IIF(v=c,'+', ' ')
ON SELECTION POPUP kadr REPLACE v WITH IIF(v=c,' ',c)
DO dat
ACTIVATE POPUP kadr
```

```
PROCEDURE dat      && Процедура формирования строки дата+время
c=DTOC(DATE(),1)+LEFT(TIME(),2)+SUBSTR(TIME(),4,2)+RIGHT(TIME(),2)
RETURN
```

В самом меню выясняется, каково значение поля V. Если оно не равно C, значит, данное поле не отобрано, если равно, справа появляется плюс. Для работы такого меню не требуется очистка поля V, поскольку каждая *дата+время* активации меню будет заведомо больше любого из уже имеющихся значений V. Неудобством, конечно, является необходимость выделения четырнадцати байт под поле V.

Размер этого поля можно сократить до одиннадцати байт, если отказаться от первых трех цифр года. Тогда повторных дат не будет до конца десятилетия, что является совершенно достаточным сроком для жизни практически любой системы (от 1993 г.).

Радикально уменьшить размер этого поля можно, если хранить в нем не абсолютную, а относительную *дату+время*, а также если перейти к двоичному ее кодированию. Хотя FoxPro и не имеет битовых полей, можно имитировать их с помощью символьных строк. Для этого исходное число по известному алгоритму преобразования систем счисления последовательно делится на новое основание — число 256.

В качестве точки отсчета следует взять наибольшую дату, которая будет заведомо меньше любой даты в будущем. Этой датой может быть дата начала разработки прикладной системы, например 01.01.1993 г. Программа приведена ниже.

```
USE kadr
Z=' '
DEFINE POPUP kadr PROMPT FIELD fam+IIF(v=z,'+', ' ')
ON SELECTION POPUP kadr REPLACE v WITH IIF(v=z,' ',z)
```



```
DO preobr
ACTIVATE POPUP kadr
```

```
PROCEDURE preobr      &&-----Процедура двоичного преобразования
* c= количество секунд от 1.1.1993г. до вызова меню
c=(DATE()-{01.01.1993})*86400+INT(SECONDS())
DO WHILE c>=256
    z=CHR(MOD(c,256))+z      && Формирование числа из остатков
    c=INT(c/256)             && Новое значение C
ENDDO
z=CHR(c)+z                  && Итоговая строка
RETURN
```

Здесь в процедуре PREOBR в переменной C формируется число секунд, прошедших от исходной даты. Оно вычисляется как число прошедших дней, умноженных на число секунд в сутках, плюс число секунд, прошедших от начала суток. Далее значение переменной C делится на 256. Остатки от делений (MOD(C,256)) и последнее частное, преобразованные в строковую форму, как раз и образуют искомую цепочку символов Z, которая в дальнейшем заносится в поле V.

Максимальное число в "двоично-байтовом" представлении определяется формулой 256^{**N-1} , где N — число байт в поле. Таким образом, если взять N=4, "емкости" поля хватит более чем на сто лет.

Один раз перед началом работы с базой значения поле V должно быть очищено. В данном случае должно быть заполнено символами CHR(0), т.е.

```
REPLACE ALL v WITH CHR(0)+CHR(0)+CHR(0)+CHR(0)
```

Удобно также использовать индексирование. Ниже приведена программа отбора записей с помощью POPUP-меню. Нажатием клавиши Enter делается выбор, а нажатием клавиши Space — завершение отбора и обработка записей. Предварительно база KADR.DBF индексируется по логическому полю V командой INDEX ON v TO vibor COMPACT.

```
USE kadr INDEX vibor ORDER 0
DEFINE POPUP kadr PROMPT FIELD fam+IIF(v,'+', ' ');
    TITLE 'Отбор' Enter 'Обработка' Space
ON SELECTION POPUP kadr DO vibor
ACTIVATE POPUP kadr
REPLACE v WITH .f. FOR v && Сброс поля V
USE
```

```
PROCEDURE vibor      &&----- Процедура выбора и обработки
DO CASE
CASE LASTKEY()=32      && Нажатие клавиши Space
    HIDE POPUP kadr      && Скрытие POPUP-меню
    SCAN FOR v           && Сканирование базы <обработка
    ENDSCAN              && выбранных записей>
    DEACTIVATE POPUP      && Удаление меню
CASE LASTKEY()=13      && Нажатие клавиши Enter
    REPLACE v WITH !v    && Инверсия поля V
ENDCASE
RETURN
```

Работа программы очевидна. Необходимо только отметить, что здесь при поисковых операциях с индексами всюду используется технология Rushmore, и поэтому индекс открыт с опцией ORDER 0. Это позволяет исключительно быстро выполнить команду очистки поля V — REPLACE v WITH .f. FOR v. Очевидная альтернатива — пара команд SEEK .t. и REPLACE v WITH .f. WHILE v при SET ORDER TO 1 здесь не будет работать, поскольку командой REPLACE изменится ключевое поле индекса. При этом сразу после замены .t. на .f. WHILE-условие перестает быть истинным и команда REPLACE завершается. Проверка показывает, что команда с опцией FOR здесь работает правильно.

С изменением некоторых деталей индексирование может быть использовано и с полем V любого другого типа.

В случае, если мы не имеем возможности включить в базу данных дополнительное поле (аналогичное полю V), можно использовать массив переменных, в который будут заноситься номера отобранных записей. Очевидно, возможность отбора данных ограничена размерностью массива и доступной основной памятью.

Пр и м е р. Пусть используется массив A(50), т.е. разрешен отбор до пятидесяти записей. Из базы KADR.DBF организовано POPUP-меню также с именем KADR, которое кроме поля FAM содержит признак уже сделанного выбора — знак "+" (если выбор был произведен). Этот символ присутствует в меню только в случае, если в массиве A имеется элемент, содержащий номер записи, на которой находится курсор меню (IF(ASCAN(a,RECNO())#0,'+', ' ')).

```

CLEAR
SET TALK OFF
USE kadr
DIMENSION a(50)                                && Описание массива A
a=0                                              && Обнуление массива
DEFINE POPUP kadr PROMPT FIELD kadr.fam+;
      IIF(ASCAN(a,RECNO())#0,'+', ' ') ;
      TITLE 'Отбор работников (до 50)' FOOTER 'Завершение - Space'
ON SELECTION POPUP kadr DO otbor
ACTIVATE POPUP kadr

PROCEDURE otbor      &&----- Процедура обработки выбора
DO CASE
CASE LASTKEY()==13      && Если клавишей Enter
      i=ASCAN(a,RECNO()) && Поиск в массиве номера этой записи
      IF i#0             && Если найден,
          a(i)=0         && элемент обнуляется
      ELSE               && Иначе
          i=ASCAN(a,0)   && ведется поиск пустого элемента
          IF i=0         && Если весь массив заполнен,
              WAIT "Отбор закончен" WINDOW NOWAIT && сообщение
              RETURN     && и выход
          ENDIF
          a(i)=RECNO()   && В свободный элемент заносится номер записи
      ENDIF
CASE LASTKEY()==32      && Если клавишей Space
      =ASORT(a)          && Сортировка
      FOR i=1 TO 50      && Просмотр всех элементов массива
          IF a(i)#0      && Если он не пуст, указатель
              GO a(i)    && записей перемещается на нужный
              ?fam,tab,recno() && номер и печатается запись из БД
          ENDIF
      ENDFOR
      DEACTIVATE POPUP
ENDCASE
RETURN

```

Клавиши выбора Enter и Space обрабатываются в процедуре OTBOR. В нашем случае за клавишей Enter оставлены обычные функции выбора, а клавиша Space будет завершать процесс отбора.

Итак, если нажата Enter (LASTKEY()==13), в массиве A ищется элемент со значением, совпадающим с номером отобранной записи (ASCAN(a,RECNO())). Если такой элемент находится, значит, еще ранее эта запись была выбрана. Этот факт расценивается программой как желание пользователя отменить свой выбор (такой элемент получает значение 0, что влечет удаление символа "+" из меню). В противном случае ведется поиск первого свободного элемента в массиве A (ASCAN(a,0)), в который и заносится номер отобранной записи. Если такой элемент не найден, значит, весь массив уже заполнен. Об этом делается сообщение, и процедура покидается.

Используя заполненный массив A, можно очень быстро отобрать нужные

записи в базе, не прибегая к ее сплошному перебору.

Так, при нажатии клавиши Space (LASTKEY()=32) просматриваются все элементы массива. Если очередной элемент не пуст, указатель записей устанавливается на запись базы с номером, содержащимся в переменной A(I), и из нее на экран выводятся фамилия и табельный номер. Чтобы база данных просматривалась в одном направлении, массив A предварительно сортируется (=ASORT(a)).

Совмещение светового и клавишного меню. Необходимость совмещать клавишное и световое меню может возникать довольно часто. При этом средствами светового меню обычно выбирается объект обработки, а клавишное меню позволяет указать одно из возможных действий над ним. Такая ситуация может быть типичной, например, для POPUP-меню, содержащего имена файлов.

Выбор в POPUP-меню осуществляется нажатием клавиш Space (код клавиши 32) и Enter (13), а выход с деактивацией меню — клавиши Escape (27) или клавиш с горизонтальными левой (19) и правой (4) стрелками. Кроме того, как выяснилось, выбор возможен также нажатием клавиш Ctrl-Enter (10) и Ctrl-End/W (23), а выход — Tab (9) и Shift-Tab (15). Таким образом, для выбора можно использовать 4 клавиши, а для выхода — 5. Нажатие каждой такой клавиши далее выявляется функцией LASTKEY() и обрабатывается нужным образом. Однако четырех клавиш выбора может не хватить. Кроме того, следует учесть, что, хотя возможность выбора пункта POPUP-меню нажатием клавиш Ctrl-Enter и Ctrl-End/W существует, она не указана в документации. Ввиду этого в следующих версиях эта возможность может быть и утрачена.

Здесь можно воспользоваться командой SET FUNCTION TO, которой закрепим за F-клавишей код клавиши Enter (символ ";", или CHR(13), или '{Enter}'). Это позволит нам "обмануть" меню. Нажатие на подобную F-клавишу воспримется POPUP-меню, как нажатие на Enter, но функция LASTKEY() распознает фактически использованную клавишу.

Кроме того, используя команду ON KEY LABEL, можно сделать назначения любым, а не только F-клавишам. При этом, однако, после завершения процедуры обработки клавишного выбора курсор в POPUP-меню со списком файлов не остается в выбранной позиции меню, а устанавливается на его первую строку. Это может быть неудобно, и поэтому, вероятно, имеет смысл пользоваться такой командой в ситуациях, когда возврат в меню более не нужен.

Ниже приведен фрагмент программы. Здесь световое POPUP-меню BAZA предъявляет список из всех имен DBF-файлов, начинающихся на букву "K". MESSAGE-строка сообщает о возможных действиях по обработке данных. Клавиши Enter, Space, F3 и F4 образуют клавишное меню (просмотр данных, печать данных, вывод даты обновления базы, вывод размера базы данных), которое реализуется в процедуре VIBOR. Клавиша F5 (вывод структуры базы данных) обрабатывается в процедуре STRUC.

```

SET FUNCTION f3 TO ';' && Назначения клавиш
SET FUNCTION f4 TO '{Enter}'
ON KEY LABEL f5 DO struc
DEFINE POPUP baza FROM 2,14 TO 21,23 PROMPT FILES LIKE *.dbf MESSAGE;
      'Enter-просмотр, Space-печать, F3-дата, F4-размер, F5-структура'
ON SELECTION POPUP baza DO vibor
ACTIVATE POPUP baza && Активация меню

PROCEDURE vibor &&-----Обработка клавишного выбора
CLEAR
USE (PROMPT()) && Открытие файла
DO CASE && Разбор нажатий

```



```

CASE LASTKEY()=13          && Нажата клавиша Enter
  HIDE POPUP baza          && Временное освобожд. экрана от меню
  DISPLAY ALL              && Просмотр файла
  WAIT 'Просмотр закончен' WINDOW
  SHOW POPUP baza          && Восстановление POPUP-меню
CASE LASTKEY()=32          && Нажата клавиша Space
  LIST TO PRINT            && Печать
CASE LASTKEY()=-2          && Нажата клавиша F3 - вывод даты
  WAIT 'Дата обновления БД - ' + DTC(LUPDATE()) WINDOW NOWAIT
CASE LASTKEY()=-3          && Нажата клавиша F4 - вывод размера БД
  =ADIR(a, (PROMPT()))     && Получение параметров файла
  WAIT 'Размер базы данных - ' + STR(a(2)) + 'байт' WINDOW NOWAIT
ENDCASE
USE                         && Закрытие файла
RETURN

PROCEDURE struc            &&-----Обработка клавиши F5
CLEAR
USE (PROMPT())             && Открытие файла
MOVE POPUP baza TO 2,70    && Перемещение меню
LIST STRUCTURE             && Просмотр структуры базы данных
USE
WAIT ..
MOVE POPUP baza TO 2,14    && Возвращение меню
RETURN

```

Поскольку работа программы требует вывода некоторых данных на экран, здесь предусмотрено временное освобождение его от самого меню. В процедуре VIBOR команды HIDE/SHOW POPUP удаляют/восстанавливают меню на экране, а в процедуре STRUC команды MOVE POPUP отодвигают меню в свободную правую часть экрана, а после просмотра информации возвращают его на место. Команда SHOW POPUP, к сожалению, при восстановлении меню предьявляет курсор не в исходном положении.

При работе с POPUP-меню, созданным из записей базы данных, прибегать к командам SET FUNCTION нет нужды, поскольку команды ON KEY LABEL <клавиша> DO <процедура> здесь прекрасно работают и возвращают нас после выполнения <процедуры> в исходное состояние. Кроме того, при работе с таким меню фиксировать текущее положение курсора можно не только функцией PROMPT(), но и функцией RECNO(), определяющей номер записи.

BROWSE-меню. При необходимости в качестве меню можно использовать BROWSE-окно. Поскольку оно имеет облик меню, его удобно применять, когда стандартное POPUP-меню нас не удовлетворяет. Например, в случае, если база, из которой создается меню, может иметь больше 32767 записей или если нам нужно построить меню из двух сцепленных баз данных, когда одному пункту из основной базы может соответствовать несколько пунктов из вспомогательной. Сами POPUP-меню команду SET RELATION игнорируют.

Следующий пример по целям и функциям аналогичен одному из предыдущих примеров множественного выбора из базы KADR.DBF с логическим полем V.

```

USE kadr
ON KEY LABEL enter REPLACE v WITH !v          && Назначение клавиш
BROWSE TITLE 'Выбор клавишей ENTER';
FIELD fam :H='Сотрудники';
      vibor=IIF(v, '+', '') :H='Выбраны';
COLOR SCHEME 10 NODELETE NOEDIT NOAPPEND

```

BROWSE-окно в отличие от POPUP-меню не имеет никаких ограничений по управлению из него данными, однако загружается оно гораздо дольше.

Пример программы. Рассмотрим простую задачу — программу ведения базы данных TAB.DBF по вводу и редактированию табельной ведомости

подразделения. В этом примере используются многие из уже рассмотренных средств, в том числе клавишное меню. Более систематизированный разбор техники программирования будет осуществлен позже.

База данных TAB.DBF имеет следующую структуру: символьное поле фамилий (FAM), 31 числовое поле (по максимальному количеству дней в месяце), в которые заносится число отработанных человеком часов (D1, D2, D3,... D31), и числовое поле (VS), где вычисляется и хранится общая сумма всех отработанных в месяце часов.

Просто ведение табеля никаких проблем не вызывает. Для этого достаточно одной команды BROWSE. Мы, однако, постараемся создать пользователю возможно больше удобств. Во-первых, заголовок каждой колонки окна редактирования будет содержать число и день недели. Во-вторых, с тем чтобы пользователь не вводил каждый раз повторяющиеся данные (ведь большинство работников имеет стабильное число часов работы каждый день недели), в программе предусмотрены две специальные клавиши F3 и Ctrl-F3. При нажатии F3 в текущее поле вводится нормативное число рабочих часов в зависимости от числа месяца. При нажатии Ctrl-F3 все поля записи заполняются нормативными часами. В-третьих, выполняется автоматическое горизонтальное суммирование отработанных часов. И, в-четвертых, здесь предусмотрено одно вычисляемое поле Р ("Переработка"), в котором возникает предупреждение "Переработка", если число отработанных в данном месяце часов превышает принятую на данном производстве санитарную норму. Программа приведена ниже.

```
*-----Программа заполнения табеля TAB.PRГ (база TAB.DBF)-----
*----- (ввод отработанных часов для каждого дня месяца)-----
SET DATE GERMAN
SET TALK OFF
CLEAR
USE tab
DIMENSION a(7),d(31)      && Массив заголовков и массив отработок
a(1)=' Вc'                && Названия дней недели
a(2)=' Пн'
a(3)=' Вт'
a(4)=' Ср'
a(5)=' Чт'
a(6)=' Пт'
a(7)=' Сб'
d=0
@ 1,1 SAY 'Введите Месяц' GET m DEFAULT 0 RANGE 1,12 PICTURE '##'
@ 1,$+2 SAY 'и год' GET g DEFAULT 0 RANGE 92,99 PICTURE '99'
READ
dt=CTOD('01.'+STR(m,2)+'.'+STR(g,2))
kdm=DAY(GOMONTH(dt,1))-DAY(GOMONTH(dt,1))) && Число дней в месяце
n1d=DOW(dt) && Номер дня недели первого числа месяца
x=''
FOR i=29 TO kdm && Формирование строки из имен полей, если
  x=x+'d'+LTRIM(STR(i)) && дней в месяце больше 28
ENDFOR
@ 6,28 SAY 'Нормы отработки (часы)'
@ 16,28 SAY 'Отказ от ввода Esc'
@ 7,7 TO 15,72 DOUBL
FOR i= 1 TO 7
  @ i+7,9 SAY a(i)+'|' COLOR n/w
ENDFOR
k=n1d
l=1
FOR i=1 TO 6 && Число месяца
  FOR j=k TO 7 && Перебор недель месяца
    && Перебор дней недели
    x=IIF(j=1.OR.j=7,'w/r',IIF(j=6,'w/g','')) && Цвет
    @ j+7,i*10+4 SAY l PICTURE '99' GET d(l);
    PICTURE '99.99' COLOR (x)
    IF l>=kdm && Если число >= количества дней в месяце,
      i=7 && I делается заведомо больше возможного
      EXIT && и выход из внутреннего цикла
    ELSE && Иначе
```



```

        l=l+1                && число месяца увеличивается на единицу
    ENDIF
ENDFOR
k=1
ENDFOR
READ
f=''
IF LASTKEY()#27             && Если не нажата Escape,
    ON KEY LABEL f3 D0 zap1  && назначение клавиш
    ON KEY LABEL ctrl+f3 D0 zap
    f=' (F3/^F3   ввод часов)'
ENDIF
br=''                        && Инициация переменной для заголовка BROWSE-окна
j=n1d
FOR i=1 TO kdm              && Создание заголовка и переменных Р и V
    br=br+'d'+LTRIM(STR(i))+[:h=']]+STR(i,2)+a(j)+['']+','
    j=IIF(j=7,1,j+1)
ENDFOR
br1=LEFT(br,240)            && Разбиение переменной BR на две
br2=SUBSTR(br,241)          && меньшего размера
BROWSE TITLE 'Табель за'+STR(MONTH(dt),3)+'-й месяц'+;
                        STR(YEAR(dt),5)+'г '+f;
    FIELD   fam :H='Фамилия', &br1&br2;
           vs  :H='Всего' :V=sum() :F;
           p=IIF(vs>200,'Переработка','') :H='Переработка'

FUNCTION sum                &&-----Функция суммирования по горизонтали
IF LASTKEY()=13             && Если нажата Enter,
    ss=d1+d2+d3+d4+d5+d6+d7+d8+d9+d10+d11+d12+d13+d14+d15+d16+;
    d17+d18+d19+d20+d21+d22+d23+d24+d25+d26+d27+d28
    FOR i=29 TO kdm
        ss=ss+EVALUATE(FIELD(i+1))
    ENDFOR
    REPLACE vs WITH ss       && вычисляется сумма
ENDIF
RETURN

PROCEDURE zap               &&-----Процедура заполнения всей записи
FOR l=1 TO kdm              && Перебор полей
    k=FIELD(l+1)             && Имя очередного поля
    REPLACE (k) WITH d(l)    && Заполнение поля
ENDFOR
RETURN

PROCEDURE zap1              &&-----Процедура заполнения поля
k=VARREAD()                 && Имя поля
IF k='D'                     && Если поле начинается на D,
    REPLACE (k) WITH d(VAL(SUBSTR(k,2))) && заполнение поля
ENDIF
CLEAR TYPEAHEAD              && Очистка буфера
KEYBOARD CHR(13)             && Имитируется нажатие Enter
RETURN

```

Здесь сначала формируется символьный массив A(7), в который заносятся будущие элементы заголовков колонок — сокращенные названия дней недели. Выходные (суббота и воскресенье) еще выделены звездочкой. Кроме того, резервируется массив отработанных часов по дням месяца (D(31)). Первоначально он обнуляется (D=0).

Далее пользователь вводит текущий месяц (M) и год (G), что нужно для настройки вида BROWSE-окна на заданный месяц. Получая эти значения, программа формирует переменную DT (уже типа дата) для первого числа данного месяца, вычисляет фактическое количество дней в месяце (KDM) и определяет номер дня недели, приходящийся на первое число (N1D).

Затем строится удобный экран ввода нормативных часов отработки для каждого числа месяца (рис.18.12).

В самой левой колонке перечислены названия дней недели. Остальные колонки содержат числа месяца (1, 2, 3,...) и число отработанных часов (сейчас 0.00). Расположение чисел месяца и их количество соответствуют фактическим для данного месяца.

Нормы отработки (часы)

*Вс		2 0.00	30 0.00
Пн		3 0.00	31 0.00
*Сб	1 0.00	8 0.00	

Рис.18.12

В цикле (FOR i= 1 TO 7) сначала выводится столбец дней недели, а затем построчно (FOR i=1 TO 6) командой @ j+7,i*10+4 SAY...GET — числа месяца (L) и поля (D(L)) для ввода часов. "Нестандартные" дни — воскресенье, суббота и пятница выделяются цветом. В зависимости от дня недели переменная C получает значения ",W+/R" и ",W+/G" соответственно. В случае, если анализируемый день не относится к перечисленным, она получает пустое значение (""). Далее переменная C (указывается в скобках) используется в опции COLOR команды @...GET для управления цветами. Поскольку раскраска GET-поля определяется второй цветовой парой, перед символами цвета стоит запятая. "Пустой" цвет соответствует цвету по умолчанию.

Пользователь может и не вводить нормативные часы, например если они мало повторяются в данном месяце. В этом случае следует нажать клавишу Escape. Если выход из окна редактирования произошел другим образом (LASTKEY()#27), производится закрепление клавиш F3 и Ctrl-F3 за необходимыми процедурами. Кроме того, в переменной F формируется строка, содержащая указания на эти клавиши. Она будет включена в заголовок BROWSE-окна.

Далее в цикле FOR I=1 TO KDM создается переменная BR, содержащая будущие заголовки и наполнение основных колонок BROWSE-окна. Как это происходит?

BROWSE-окно для колонок дней должно получить, например следующий заголовок на август 1992 г.:

```
1*Сб| 2*Вс| 3 Пн| 4 Вт| 5 Ср| 6 Чт| 7 Пт| 8*Сб| 9*Вс ...
```

где дни недели стоят рядом с нужными числами месяца. Видимому заголовку соответствует следующая строка в команде BROWSE:

```
d1 :h='1*Сб', d2 :h='2*Вс', d3 :h='3 Пн', d4 :h='4 Вт'
```

и т.д.

Таким образом, в каждом цикле к переменной BR должны присоединяться: строка, содержащая букву D; номер очередного поля, преобразованный в строковую форму и без ведущих пробелов (LTRIM(STR(i))); буква H с двоеточием, знаком равно и апострофом для заголовка ([h=']), номер числа месяца в символьной форме (STR(i,2)); название дня недели (a(j)); апостроф, замыкающий заголовок колонки ([']), и запятая, чтобы можно было присоединить описание следующих полей. Ввиду того что апострофы здесь являются компонентами самой команды BROWSE, для включения их в текстовую константу используется другой символ-ограничитель строковых констант — прямоугольные скобки.

Важным является правильное соединение числа месяца I и дня недели J. Для этого в цикле FOR, где перебираются все числа месяца от 1 до KDM, переменная J изменяется от дня недели, приходящегося на первое число N1D, до семи. Как только J достигает 7, оно устанавливается в 1, снова растет до 7 и т.д. (j=IIF(j=7,1,j+1)).

По завершении цикла мы получим переменную BR, содержащую все заголовки полей от D1 до D31 (или меньше, если дней в месяце меньше). Размер такой строки может оказаться длиннее разрешенной для макроподстановки. Поэтому она разбивается на две (240 символов и

остальное) меньшего размера.

В самой команде BROWSE формируется заголовок BROWSE-окна, содержащий месяц и год, выводятся поле фамилий и поля дней, а также поле VS ("Всего"). Все поля дней формируются строкой из двух макроподстановок (&br1&br2), которые были получены ранее. В поле VS с помощью функции SUM() выполняется горизонтальное суммирование значений полей, содержащих отработанные часы. Чтобы расчет выполнялся всегда, опция V усилена ключом F. Вместе с тем чтобы избежать ненужных вычислений при каждом попадании курсора в поле VS, в самой процедуре сделано так, чтобы она "работала" только при нажатии клавиши Enter (IF LASTKEY()=13).

В переменной SS суммируются все D-поля от D1 до D28 включительно. Кроме того, если дней в месяце больше 28, остальные дни подсуммируются в цикле FOR i=29 TO kdm. Так как поле D1 в базе имеет номер 2 (самым первым является поле FAM), все номера здесь смещены (I+1) на единицу — EVALUATE(FIELD(i+1)). Затем SS заносится в поле VS.

Самое последнее поле, определенное в команде BROWSE, — вычисляемое поле P, в котором появляется слово "Переработка", если человеком выработано больше 200 часов (например, санитарная норма для данного производства).

Нажатие клавиши F3 вызывает процедуру ZAP1 заполнения одного текущего поля базы. В процедуре сначала запоминается имя поля (K) на котором произошло нажатие F3. Если имя поля начинается на букву D, значит, можно делать присваивание (REPLACE). Последняя команда процедуры (KEYBOARD CHR(13)) генерирует нажатие клавиши Enter, что позволяет сразу перейти на следующее поле. Таким образом, хотя с помощью клавиши F3 заполняется только одно текущее поле, ее удержание в нажатом состоянии обеспечивает ввод нужных данных и во все следующие поля базы, относящиеся к отработанным часам. Чтобы не переполнялся буфер клавиатуры (при этом некоторые поля могут быть пропущены), перед каждой генерацией кода Enter он очищается командой CLEAR TYPEAHEAD.

Процедура заполнения всей записи целиком ZAP устроена похожим образом. Здесь в цикле последовательно перебираются все поля от D1 до последнего и заполняются нужными числами. Имена полей по их номерам в базе вырабатывает функция FIELD().

Глава 19. ИЗОБРАЗИТЕЛЬНЫЕ СРЕДСТВА СУБД

Разработчик прикладных систем обработки данных должен заботиться не только о принципиально правильном функционировании программы, но и об удобстве и надежности работы с системой пользователя. Ясный и эстетичный облик экранов системы не только доставляет удовлетворение оператору, но и снижает утомляемость при работе с ней.

Эффектное оформление способствует рекламе, продаже и распространению разработанных программных продуктов, а кроме того, обеспечивает нужный "показ-эффект" при взаимодействии с начальством и заказчиками.

Использование элементов псевдографики

FoxPro содержит средства создания изображений из элементов так называемой псевдографики, т.е. примитивных графических символов, для вывода которых не нужен графический режим терминала.

■ @ <Y1,X1,Y2,X2> BOX <вырС>

Команда позволяет формировать в заданных координатах рамку и фон прямоугольника из любых символов, которые включены в <вырС>. Это символьное выражение может содержать до 9 символов (по четыре символа для углов и сторон прямоугольника, начиная с левого верхнего угла по часовой стрелке (рис.19.1)). Девятый символ (если есть) заполняет прямоугольник. Если <вырС> отсутствует, прямоугольник будет ограничен одинарной линией, если <вырС> состоит только из одного символа, он будет ограничивающим для всего прямоугольника.

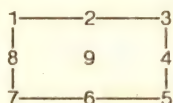


Рис.19.1

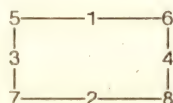


Рис.19.2

В <вырС> элементы рамки указываются в виде символьной переменной или непосредственно в апострофах, например

```
@ 5,20,12,60 BOX '++++++'
```

Команда изобразит пустой прямоугольник, очерченный звездочками, в углах которого стоят знаки "+".

■ @ <Y1,X1> TO <Y2,X2> [DOUBLE/PANEL/<окантовка>] [COLOR <список цветовых пар>]/COLOR SCHEME <вырN>]

Эта команда формирует рамку с координатами Y1,X1 и Y2,X2, а также определяет ее цвет и форму. При Y1=Y2 или X1=X2 будут изображены соответственно горизонтальная и вертикальная линии.

Рисование возможно только вниз и вправо (должны выполняться условия Y2>=Y1 и X2>=X1).

Параметры формы окантовки (по умолчанию одинарная линия):

DOUBL — рамка ограничена двойной линией;

PANEL — рамка сплошная (состоит из символов заполнения);

<Окантовка> — перечень символов окантовки, перечисленных через запятую в порядке, соответствующем рис.19.2. Если <окантовка> состоит только

из одного символа, им и будет ограничен прямоугольник. Отсутствующие элементы окантовки будут заменены на символы по умолчанию.

С помощью опции COLOR можно установить желаемый цвет рамки, не изменяя цветов по умолчанию. Подробно вопрос управления цветами разбирается дальше.

П р и м е р. Прямоугольник, ограниченный знаками '*' и '+'

```
@ 5,20 TO 12,60 '*', '*', '*', '*', '+', '+', '+', '+'
```

■ SET BORDER TO [SINGLE/DOUBLE/PANEL/NONE;
/<окантовка1> [<окантовка2>]

Команда задает форму бордюра-окантовки для рамок, меню и окон, которые будут далее использоваться по умолчанию. Рамки создаются командой @...TO, меню — командами DEFINE MENU и DEFINE POPUP, окна — командой DEFINE WINDOW.

Параметры команды определяют следующие формы окантовки:

PANEL — рамка сплошная;

NONE — окантовка не выводится;

DOUBLE — двойная линия;

SINGLE — одинарная линия (действует по умолчанию);

<Окантовка1> — собственные символы окантовки;

<Окантовка2> — то же, но для неактивных окон.

Элементы окантовки должны иметь тот же порядок и форму, которые указаны для опции <окантовка> команды @ ... TO

Кроме этого, всегда имеется возможность создавать некоторые изображения вручную. Так, целесообразно название прикладной системы в ее заголовном кадре нарисовать крупными буквами высотой в несколько строк. Каждая строка такого заголовка формируется, например, командой ? или @...SAY или внутри команд TEXT ... ENDTEXT, где элементы букв изображаются средствами псевдографики. Лучше всего для этого использовать контрастные символы (в "альтернативной" таблице ASCII-кодов это коды 219 — 223, 254). Элементы псевдографики не имеют соответствующих клавиш на клавиатуре компьютера, но могут быть сгенерированы одновременным нажатием клавиши Alt и набором ASCII-кода нужного символа на цифровой клавиатуре, т.е. Alt+ASCII-код

Удобным инструментом для работы с псевдографикой является использование системных средств FoxPro (пункты главного меню: SYSTEM+SPECIAL CHARACTERS).

Если требуется создать более сложные или динамические изображения, необходимо их программировать.

Управление цветом

В FoxPro имеются средства как непосредственного указания цветов в командах ввода-вывода, описания окон, меню, так и определения их по умолчанию. При этом приняты следующие обозначения цветов (допускаются как прописные, так и строчные буквы):

черный — N	зеленый — G	коричневый — GR
синий — B	бирюзовый — BG	красный — R
желтый — GR+	лиловый — RB	белый — W.

После символа цвета также можно указать звездочку "*" для мерцания и "+" для повышенной яркости ("+" действует только на цвет текста и не

действует на цвет фона).

В случае монохромного терминала, который допускает только белый (W) и черный цвета (N), возможно управление яркостью изображения (+), мерцанием (*), подчеркиванием (U), а также выводом текста в инверсном виде (I).

Следующая команда дает возможность управлять цветовой гаммой и интенсивностью изображения элементов текущего экрана/окна:

■ SET COLOR TO [<стандартный>
[,<дополнительный>] [,<рамка>] [,<фон>]]

Команда позволяет устанавливать цвета:

<стандартный> — цвет основных текстовых сообщений;

<дополнительный> — цвет дополнительных сообщений. К ним относятся, например, области ввода GET, выбранные пункты меню и др.;

<рамка> — цвет рамки электронно-лучевой трубки дисплея за пределами используемой области;

<фон> — цвет фона. Этот параметр используется в компьютерах, которые не позволяют установить различный фон для "стандартного" и "дополнительного" сообщений.

Раскраска стандартного и дополнительного текстов может иметь по два параметра — цвет текста и цвет фона, которые разделяются косой чертой и называются ЦВЕТОВОЙ ПАРОЙ. <Рамка> (для цветных мониторов) и <фон> (если есть) раскрашиваются только одним цветом. Например, команда

SET COLOR TO GR+/B, W+/R*, B

устанавливает следующие цвета: желтые буквы на синем фоне для основного изображения текста; ярко-белые буквы на красном для дополнительного (при этом текст будет мерцать); синий цвет рамки дисплея.

Команда SET COLOR TO без параметра осуществляет переход к цветам, установленным в FoxPro по умолчанию.

Рассмотренная команда хорошо известна программистам, работающим на любом из dBASE-языков. Однако для многих экранных объектов FoxPro двух цветовых пар (основной и дополнительной) совершенно недостаточно. Ввиду этого разработчики FoxPro предлагают более мощные средства управления цветами, ориентированные на понятие ЦВЕТОВОЙ СХЕМЫ (с номерами от 1-го до 24-го), которая состоит из СПИСКА ЦВЕТОВЫХ ПАР. Список цветовых пар содержит пары цветов (до десяти), разделенные запятыми. Теперь цвет большинства объектов может быть определен в индивидуальном порядке либо ссылкой на номер используемой цветовой схемы (COLOR SCHEME <вырN>), либо непосредственным перечислением цветовых пар (COLOR ...).

Каждая цветовая пара в списке цветовых пар указывает цвет определенного элемента объекта (окна, меню и т.д.).

Цвета в цветовой схеме могут быть установлены двумя способами — в диалоге или командой SET COLOR OF SCHEME.

В первом случае надо выбрать пункт WINDOW в главном меню системы, а в нем — пункт COLOR. После этого мы попадем в окно Установщика Цвета (Color Picker). В его правом верхнем углу мы увидим имя текущего объекта, например

Window

который можем раскрасить по своему усмотрению. Если мы переместим

курсор или маркер мыши в этот прямоугольник, то увидим перечень всех 24 цветовых схем, из которых первые двенадцать указаны именами соответствующих объектов (1: *Usr Winds* — пользовательские окна; 2: *Usr Menu* — пользовательские меню; 3: *Menu Bar* — BAR-меню и т. д.), а остальные двенадцать — номерами. В левой части экрана отображен макет раскраски, где каждый элемент объекта назван по имени и указан номер его цветовой пары.

Здесь мы рассмотрим только элементы окраски BROWSE-окна. Его цветовая схема (схема 10) имеет следующий вид:

W+/BG, GR+/B, GR+/W, GR+/W, N+/W, GR+/GR, W+/B, N+/N, GR+/W, N+/W, +

Ниже по порядку перечислены используемые командой цветové пары и указаны окрашиваемые ими элементы.

- 1 — невыделенные записи (W+/BG);
- 2 — выделенное поле — поле текущей записи, в котором находится курсор (GR+/B);
- 3 — бордюр, включая знаки-маркеры на бордюре (GR+/W);
- 4 — заголовок активного Browse-окна (GR+/W);
- 5 — заголовок неактивного Browse-окна (N+/W);
- 6 — текст, выделенный в текущем поле для копирования или удаления с помощью клавиши Shift и клавиш управления курсором или мышью (GR+/GR);
- 7 — текущая запись (W+/B). Кроме того, цветом фона (B) в этой цветовой паре раскрашивается точка-пометка записи, предназначенной для удаления (нажатием клавиш Ctrl-T). Чтобы гарантировать ее видимость, цвет точки должен быть контрастным к цвету фона из пары 1 (здесь B на фоне BG — синяя точка на бирюзовом фоне);
- 8 — тень окна (N+/N).

Соответствия для других объектов можно выяснить, выбрав нужные названия в меню или обратившись к приложению.

Если нас не устраивают установленные цвета, мы легко можем изменить их желаемым образом. Однако, чтобы эти новые цвета действовали не только в текущем сеансе, но и могли быть использованы в дальнейшем, их следует сохранить, обратившись к пункту меню SAVE в Color Picker, в котором все 24 цветовые схемы (ЦВЕТОВОЙ НАБОР) запоминаются в файле FOXUSER.DBF с заданным именем (до десяти символов). При следующем старте FoxPro можно загрузить цветовой набор, использовав пункт LOAD в Color Picker.

Другая возможность — это использование команды

■ SET COLOR SET TO [<Имя цветového набора>]

которая загружает указанный цветовой набор из файла FOXUSER.DBF, определенный ранее командой SET COLOR OF SCHEME или при работе с меню. Аналогичный результат будет получен, если включить эту команду в файл конфигурации системы CONFIG.FP в форме

COLOR SET TO = <Имя цветového набора>.

Но обычно нет никакой необходимости переопределять первые двенадцать системных цветовых схем. Цветовые схемы 13, 14, 15 и 16 также лучше не трогать, поскольку разработчики пакета планируют их использовать в дальнейшем — в следующих версиях системы. Остальные восемь схем мы можем применять по собственному усмотрению.

Следующая команда позволяет, не прибегая к диалогу, установить цвета в

цветовой схеме <вырN1>, указав их в <списке цветowych пар> или скопировав из другой цветовой схемы <вырN2>.

- SET COLOR OF SCHEME <вырN1> TO;
TO [[<список пар цветов>]/[SCHEME <вырN2>]]

Допускается указывать не все 10 пар, заменяя пропущенные запятой.

Пример:

```
SET COLOR OF SCHEME 20 TO;  
W+/B,W+/R,W+/N*,G+/N*,G+/N*,W+/B*,GR+/B,N+/N,W+/B,W/B  
SET COLOR OF SCHEME 16 TO SCHEME 10  
SET COLOR OF SCHEME 6 TO W+/N,...N/BG
```

В примере устанавливается цветовая схема 20, схема 16 получает цветowe пары из схемы 10, в схеме 6 первая и четвертая пары получают новые значения.

Если команда SET COLOR OF SCHEME <вырN1> TO используется без других параметров, то цветовая схема <вырN1> получит цвета из текущего цветового набора.

Включение в файл CONFIG.FP рассмотренной команды в форме COLOR OF SCHEME <вырN> = <Список цветowych пар> также повлечет установку нужного списка цветов для указанной цветовой схемы при старте FoxPro.

Содержание списка цветowych пар для цветовой схемы с номером <вырN> всегда можно выявить с помощью функции

- SCHEME(<вырN>)

Следующая команда позволяет устанавливать цветowe пары отдельных элементов в схемах 1 (пользовательские окна) и 2 (пользовательские меню).

- SET COLOR OF NORMAL/MESSAGES/TITLES/BOX/HIGHLIGHT
/INFORMATION/FIELDS TO [<станд>[,<доп>]]

Здесь каждому поименованному элементу задаются две цветowe пары: для "стандартного" и "дополнительного" изображений. В целях совместимости команда заимствована из СУБД dBASEIII и не дает каких-либо новых возможностей.

С EGA и VGA мониторами может быть также использована команда

- SET BLINK ON/OFF

которая управляет мерцанием/яркостью изображения.

При SET BLINK ON элементы экрана, для которых установлены цвета со знаком "*", будут мерцающими. При OFF цвет фона для тех же элементов будет более ярким. Это вдвое расширяет цветовую гамму. По умолчанию SET BLINK ON.

Перечисленные команды позволяют полностью управлять установкой цветов.

Команда, рассматриваемая ниже, позволяет изменить цвет уже существующего изображения внутри прямоугольной области экрана/окна с координатами Y1,X1 и Y2,X2.

- @ <Y1,X1> FILL TO <Y2,X2>
[COLOR <список цветowych пар>/COLOR SCHEME <вырN>]

Раскрашивание возможно только вниз и вправо, т.е. должны выполняться условия $Y2 \geq Y1$ и $X2 \geq X1$. Вывод в указанную область после команды @...FILL будет иметь цвета, установленные ранее. Если цвета не указаны, команда только очистит прямоугольник.

Пример:

.SET COLOR TO w+/n	&& Установка цвета (белый на черном)
@ 10,20 SAY 'СУБД FoxPro'	&& Вывод текста
@ 5,8 FILL TO 20,40 COLOR g/w	&& Перекрашивание (зеленый на белом)
@ 12,20 SAY 'СУБД FoxPro'	&& Вывод в прежних цветах

Команда удобна для выделения некоторых элементов экрана в целях предупреждения или привлечения внимания пользователя. С ее помощью можно создавать эффектные динамичные изображения, особенно для заголовков. Так, пусть уже выведен заглавный кадр некоторой прикладной системы. Следующий фрагмент программы после паузы в 0.3 с медленно (со скоростью одна строка за 0.01 с) перекрасит весь экран в иной (N/GB) цвет:

```
FOR i=0,24
  @ i,0 FILL TO i,79 COLOR n/gb
  =INKEY(0.01)
ENDFOR
```

Применяя элементы псевдографики и/или команды управления цветом, можно легко строить столбиковые диаграммы. Пусть мы хотим расположить диаграмму горизонтально в колонках с 0-й по 78-ю. Для масштабирования выводимых данных необходимо найти максимальный элемент, используя который вычисляется коэффициент пересчета фактических значений в длины столбцов диаграммы. Этот коэффициент K определяется как отношение занимаемой ширины экрана (79 колонок) к максимальному диапазону данных:

$$K = 79/\text{MAX}$$

Далее длина каждого столбца определяется в зависимости от значения данного X по формуле

$$\text{ROUND}(K \cdot X, 0),$$

в которой длина столбца округляется до целой части.

Пример. Положим, необходимо построить совмещенную столбиковую диаграмму месячных выработок (VIR) работников бригады 6, где кроме выработки показана также и премия (30 %). При выводе каждой строки диаграммы ей должна предшествовать строка с указанием табельного номера работника TAB и выработки VIR.

Программа приведена ниже.

```
*-----Модуль построения горизонтальных диаграмм-----
SET TALK OFF
SET COLOR TO GR+/B
CLEAR
USE brig6
CALCULATE MAX(vir) TO maxv && Максимальная выработка = MAXV
K=79/(maxv+0.3*maxv) && Масштабирование выработки и премии
? 'ДИАГРАММА ВЫРАБОТКИ БРИГАДЫ' AT 20
SCAN
  ? 'Tab.-',TAB,'Выработка -',vir,'p'
  SET COLOR TO R
  ? REPLICATE('█',ROUND(K*vir,0)) && Выработка
  SET COLOR TO G
  ?? REPLICATE('█',ROUND(K*0.3*vir,0)) && Премия
  SET COLOR TO GR+/B && Восстанавливается исходный цвет
ENDSCAN
```

Здесь первоначально в базе BRIG6.DBF находится и упоминается в

переменной MAXV максимальная выработка в бригаде (CALCULATE MAX(vir) TO maxv). Затем определяется коэффициент K как отношение наибольшей возможной суммы (найденный максимум плюс 30 % премии) к ширине экрана. Далее сканируется вся база. Для каждой ее записи сначала выводится красным цветом столбец выработки, который продолжается столбцом премии зеленого цвета. Диаграмма изображена на рис.19.3.

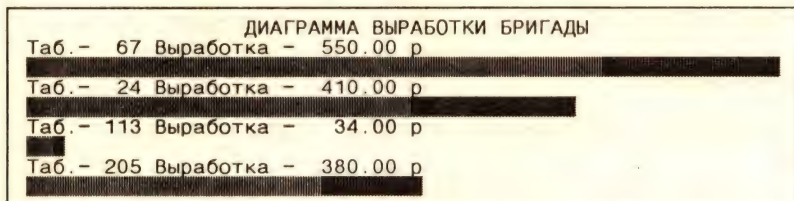


Рис.19.3

Для выдачи диаграмм на экран совершенно достаточно выделить их цветом. Если же предполагается печать, следует, как в данном случае, использовать какой-нибудь символ заполнения.

При построении диаграмм удобны также команды вывода вида @...SAY. С их помощью легко создавать и вертикальные столбиковые диаграммы. Ниже приведен фрагмент такой программы, где столбцы высотой до 20 строк, окрашенные в зеленый цвет, располагаются вверх от 23-й строки на фоне горизонтальной сетки из четырех линий. Кроме того, здесь предусмотрена возможность горизонтального скроллинга экрана, если диаграмма не умещается в его 80 колонок.

```
*-----Модуль построения вертикальных диаграмм-----
k=20/maxv          && Масштабирование по вертикали
i=1                && Номер текущей колонки экрана
@ 24,10 SAY 'Выработка бригады'
SCAN
  t=STR(tab,3)      && Сканирование базы
                    && Преобразование поля TAB в строку
  FOR j=1 TO 3      && Сканирование табельного номера
    @ j+20,i SAY SUBSTR(t,j,1) && Вывод очередной цифры табеля
  ENDFOR
  @ 4,i SAY '____'  && Горизонтальная сетка
  @ 9,i SAY '____'
  @ 14,i SAY '____'
  @ 19,i SAY '____'
  @ 23-ROUND(K*vir,0),i+1 FILL TO 23,i+2 COLOR /G && Столбец
  IF COL(<)<79      && Если текущий вывод еще в экране
    i=i+3           && Переход на три колонки вправо
  ELSE              && Иначе -
    WAIT ''         && пауза для просмотра
    SCROLL 1,0, 23,79,0,-3 && и горизонтальный скролинг
    i=i-3           && Возврат назад
  ENDIF
ENDSCAN
```

В FoxPro имеется команда установки различных режимов работы дисплея.

■ SET DISPLAY TO

CGA/COLOR/EGA25/EGA43/MONO/MONO43/VGA25/VGA50

Она переключает терминал в режим работы CGA-, EGA-, VGA- и MONO-адаптеров с 25, 43 и 50 строками. Естественно, что такая возможность должна быть поддержана техническими средствами (очевидно, что VGA-адаптер может работать в режиме CGA, но не наоборот).

Если желаемый режим (MODE) не поддерживается, выдается сообщение "Display mode not available". Выяснить тип адаптера и монитора можно с помощью функции SYS(2006). Кроме того, определить цветной или черно-белый монитор установлен на компьютере, можно функцией ISCOLOR().

Работать в режимах плотного вывода (43 и тем более 50 строк) очень утомительно. Это имеет смысл только на мониторах с большим экраном или при отладке программ, когда на экране нужно разместить окна трассировщика и отладчика, которые не накрывали бы окна, генерируемые программой.

Управление звуком

FoxPro располагает некоторыми средствами управления звуком.

Команда

■ SET BELL ON/OFF

включает/выключает звуковое сопровождение редактирования данных в командах BROWSE, CHANGE, APPEND, EDIT, INSERT, READ. При ON (по умолчанию) выход из редактируемого поля будет сопровождаться сигналом.

Команда

■ SET BELL TO <частота>,<длительность>

контролирует частоту (от 19 до 10 000 герц) и длительность (от 1 до 19 с) звукового сигнала.

Следующий фрагмент программы

```
SET BELL ON
FOR i=1 TO 5
    SET BELL TO i*80, i*3
    ??CHR(7)
ENDFOR
```

генерирует примитивную мелодию с помощью управления частотой и длительностью сигнала, выводимого командой ??CHR(7).

Глава 20. РАБОТА С ОКНАМИ

Очень важным для СУБД является возможность работы с окнами. Каждое окно — это, по существу, автономный экран системы. Одновременно может быть доступно несколько не мешающих друг другу окон, что позволяет создавать очень удобный "многослойный" пользовательский интерфейс.

Окна можно открывать, закрывать, изменять размеры и положение с помощью команд, клавиатуры и мыши. Окна могут существовать отдельно и быть вложенными друг в друга.

Различают три размера окон — нормальный, минимальный и максимальный.

Нормальный размер окна определяется при описании окна командой `DEFINE WINDOW` и может быть изменен командой `ZOOM WINDOW`.

Максимальное окно занимает весь экран или всю площадь другого "старшего" окна, в котором оно находится.

Минимальное окно стягивается в одну строку, в которой виден только его заголовок (если есть) или имя окна.

Вид окна определяется опциями команды описания окна `DEFINE WINDOW`, которая будет рассмотрена ниже. По умолчанию (необязательных опций нет) окно обрамлено одинарной рамкой и не имеет никаких средств управления.

Рассмотрим общий вид окна, которое было описано с опциями `TITLE`, `SYSTEM`, `CLOSE`, `FLOAT`, `GROW`, `ZOOM`, `MINIMIZE`. Такое окно (рис.20.1) обрамлено сплошной полосой контрастного цвета (опция `SYSTEM`), на которой содержатся важные управляющие и информационные символы. Большинство системных окон FoxPro отображается именно так.

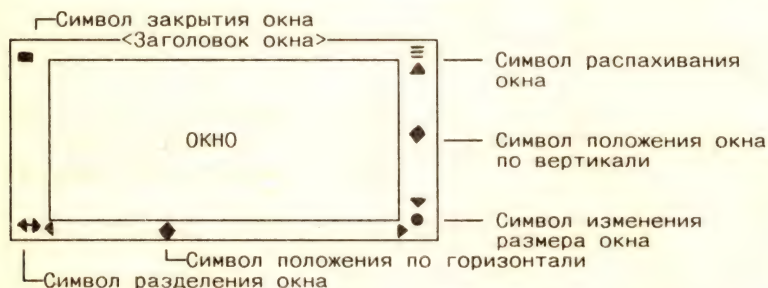


Рис.20.1

Нажатие левой кнопки мыши при совмещении ее маркера с соответствующим символом окна или использование специальных клавиш повлечет трансформацию окна.

Рассмотрим символы управления окном. В скобках указаны опции, определяющие данные возможности окна.

- — окно закрывается и удаляется с экрана. В программном режиме определение окна сохраняется, а в командном — нет (опция `CLOSE`). В общем случае с клавиатуры окно закрыто быть не может. Это не относится к окнам данных, `BROWSE/CHANGE`-окнам, а также к окнам, содержащим команду `READ`, мемо-поле или текстовый файл. В этом случае могут быть использованы клавиши `Escape` или `Ctrl-W`.
- ≡ — окно раскрывается до максимального размера. Повторный выбор этого символа мышью влечет возврат к исходному размеру окна. С клавиатуры окно раскрывается нажатием клавиш `Ctrl-F10` (опция `ZOOM`).

- — если накрыть этот символ маркером мыши, нажать и не отпускать ее кнопку, то можно "отбуксировать" нижний правый угол окна для изменения его размера. При этом рамка окна мигает (GROW). С клавиатуры режим изменения размера окна вызывается нажатием клавиш Ctrl-F8. Далее можно пользоваться клавишами управления курсором.
- ⇐ — этот символ существует только для некоторых окон и указывает на возможность разделения окна на две части. Если накрыть этот значок маркером мыши, то можно "отбуксировать" его вправо. При этом будет перемещаться и вертикальная линия разделения окна. Такая возможность существует для окна редактирования (команды BROWSE/CHANGE). Затем обе половины окна можно настроить на желательный режим работы. Это действие с клавиатуры может быть выполнено только через системное BROWSE-меню (пункт Browse+Resize Partitions). Режим разделения окна также доступен для окна отладчика DEBUG.

На верхней ограничивающей линии окна отображается <заголовок окна>, который является именем базы данных или любой строкой, указанной в качестве заголовка (TITLE) в команде описания окна.

Для перемещения всего окна нужно поставить маркер мыши на верхнюю горизонтальную границу окна, содержащую заголовок, и "отбуксировать" его в нужное место. С клавиатуры режим перемещения окна устанавливается нажатием клавиш Ctrl-F7, при этом рамка мигает (FLOAT).

Здесь же можно осуществить переход к минимальному окну, нажав два раза на кнопку мыши. Если перед этим еще нажать и удерживать клавишу Shift, то минимальное окно переместится в правый нижний угол экрана (MINIMIZE). Такой механизм позволяет пользователю организовать в одном месте экрана компактный "склад" всех нужных окон и при необходимости быстро их открывать.

С клавиатуры переход к минимальному окну осуществляется нажатием клавиш Ctrl-F9 (с перемещением в нижний угол клавишами Shift-Ctrl-F9). Возврат из минимального окна в исходное состояние выполняется теми же действиями (можно без клавиши Shift).

Кроме того, на нижней и правой границах окна могут быть показаны символы (ромбики) горизонтального/вертикального положения курсора в окне относительно соответствующих границ данных, часть которых видна в окне. Эти символы ("ползунки") также можно "буксировать" для быстрого перемещения данных в окне. Прокручивание данных в желаемом направлении может быть осуществлено и при установке мыши на символы границ данных (треугольники) и вообще на любое место правой/нижней границы окна.

На экране может быть открыто сразу несколько окон. Переход между окнами может быть выполнен последовательно клавишами Ctrl-F10 или произвольно мышью.

Замечание. Символы распаивания и изменения размера окна могут быть и другими. Это зависит от отображения кодов 240 и 249 в ASCII-таблице русского алфавита.

Практически все указанные действия могут быть выполнены программно с помощью команд управления окнами.

Каждое окно прикладной системы предварительно должно быть определено, а затем при необходимости активировано. Ненужные окна отключаются или удаляются. В процессе работы с окнами используются следующие команды:

DEFINE WINDOW — определяет окно и задает все его свойства.

ACTIVATE WINDOW — активирует окно при необходимости его

использовать.

HIDE WINDOW — делает скрытым активное окно. При этом выдачи все равно идут в это окно, но невидны на терминале.

SHOW WINDOW — снова делает видимым скрытое окно.

MOVE WINDOW и **ZOOM WINDOW** — осуществляют перемещение окна по экрану и изменяют его размер.

SAVE WINDOW — сохраняет описание и содержание окна в файле.

DEACTIVATE WINDOW — удаляет (деактивирует) временно ненужное окно с экрана. Определение окна в памяти сохраняется и может быть снова использовано для активации.

CLEAR/RELEASE WINDOWS — удаляют ненужные в данный момент окна при недостатке памяти. Повторное их использование тогда возможно только при повторном описании.

Рассмотрим "оконные" команды.

С помощью команды

```
■ DEFINE WINDOW <имя окна>
  FROM <Y1>,<X1> TO <Y2>,<X2>
  [FOOTER <вырC1>] [TITLE <вырC2>]
  [SYSTEM/DOUBLE/PANEL/NONE/<вырC3>]
  [CLOSE] [FLOAT] [GROW] [SHADOW]
  [ZOOM] [FILL <вырC4>] [MINIMIZE]
  [COLOR SCHEME <вырN>/COLOR <список цветовых пар>]
```

создается окно, дается ему <имя> и определяются его параметры:

Y1,X1 и **Y2,X2** — координаты верхнего левого и нижнего правого углов окна на экране. Эти координаты могут лежать и за пределами экрана. Допускается, чтобы размер окна в два раза превышал по числу строк и столбцов размер текущего экрана. Окна также могут помещаться одно внутри другого.

FOOTER <вырC1> и **TITLE <вырC2>** — назначение окну заголовков, вставляемых в центр нижней и верхней границ окна.

SYSTEM/DOUBLE/PANEL/NONE/<вырC3> — эти опции определяют форму границ окна:

DOUBLE — двойная линия;

NONE — граница не отображается;

PANEL — сплошная полоса;

<вырC3> — здесь можно указать свои символы определения границы. Этих символов может быть до восьми — по одному для каждой из сторон и углов окна. Порядок указания символов для элементов окантовки окна совпадает с порядком, принятым в командах **@...TO...** и **SET BORDER TO** (см. соответствующие разделы).

SYSTEM — устанавливает форму границы окна, принятую для системных окон СУБД (сплошная полоса). В ее углах показываются символы управления (изменение размеров, "распахивание" окна, удаление окна) мышью, если эти параметры **GROW**, **ZOOM**, **CLOSE** присутствуют в описании окна (см. ниже). Если параметры присутствуют без опции **SYSTEM**, то мышь действует в соответствующих местах окна, но символы управления не будут показаны.

По умолчанию окно очерчивается одинарной линией.

CLOSE — позволяет закрыть окно с помощью мыши. Закрытие окна удаляет его из памяти. Для повторного вывода окна нужно его снова определить

и активировать.

FLOAT — допускается перемещение окна с помощью клавиш Ctrl-F7 или мыши.

GROW — позволяет изменять размер окна клавишами Ctrl-F8 или мышью.

ZOOM — увеличение окна на весь экран и возвращение к прежнему размеру клавишами Ctrl-F10 или мышью.

SHADOW — окно будет отображаться с тенью.

FILL <вырC4> — указывает возможный символ-заполнитель окна.

MINIMIZE — допускается приведение окна к Минимальному размеру.

COLOR SCHEME <вырN>/COLOR <список цветовых пар> — устанавливает цвета окна непосредственно или с помощью цветовой схемы. По умолчанию окнам присваиваются цвета из COLOR SCHEME 1.

Именно с помощью команды **DEFINE WINDOW** выделяется необходимый объем основной памяти компьютера для создания окна.

Команда

■ **ACTIVATE WINDOW**

[[<окно1>] [,<окно2>] [,...]]/ALL [BOTTOM/TOP]

выводит на дисплей и активирует ранее определенное командой **DEFINE WINDOW** перечисленные или все (ALL) окна. Теперь экранные выводы будут направляться в окно, активированное последним.

Опции **BOTTOM/TOP** указывают на то, что активированные окна размещаются под/перед уже имеющимися на экране. По умолчанию **TOP**.

Команда

■ **SHOW WINDOW <окно1> [,<окно2> ...]/ALL**

**[IN [WINDOW] <окноN>/IN SCREEN]
[REFRESH] [TOP/BOTTOM/SAME]**

управляет выдачей и расположением всех (ALL) или указанных <окон> на экране (**IN SCREEN**) или внутри другого окна (**IN WINDOW**). Опция **IN SCREEN** действует по умолчанию.

Параметр **REFRESH** рекомендуется использовать для повторной выдачи окна **BROWSE**, поскольку изменения данных, сделанные в нем командой **REPLACE**, не отображаются под курсором.

Если окна определены, но не активированы или скрыты, команда их выводит на экран. Если окна выведены, команда позволяет изменить последовательность их появления относительно уже открытых окон — поверх, под или в том же положении, которое предшествовало их деактивации (**TOP/BOTTOM/SAME**).

Командой **SHOW WINDOW** нельзя указать, куда направлять вывод. Назначение окна вывода осуществляется командой **ACTIVATE WINDOW**.

Команда

■ **DEACTIVATE WINDOW <окно1>[,<окно2>] [,...]/ALL**

деактивирует перечисленные или все (ALL) активные окна и удаляет их с экрана без удаления из памяти. Теперь весь вывод будет направляться в окно, активация которого предшествовала данному окну или на экран, если окон больше нет.

Команда

■ **MOVE WINDOW <окно> TO <Y>,<X> / BY <вырN1>,<вырN2>**

перемещает <окно> на экране/окне. Допускается перемещение окна в абсолютно или относительно заданную позицию экрана. Параметры соответствуют новой позиции (Y,X) левого верхнего угла окна, или его смещению (<вырN1>,<вырN2>) относительно старой.

Если окно определено, оно может быть перемещено, в том числе и до его активации.

Команды удаления окон:

- CLEAR WINDOWS — удаляет все окна с экрана и из памяти.
- RELEASE WINDOWS <окна> — удаляет с экрана и из памяти все перечисленные <окна>.

Команда

- SAVE WINDOW <окна>/ALL TO <файл>/TO MEMO <мемо-поле>

сохраняет текущие указанные или все (ALL) <окна> в <файле> или <мемо-поле>.

Восстановление окон может быть выполнено командой

- RESTORE WINDOW

П р и м е р. Ниже приведен простой пример описания и работы с окном KADR для базы KADR:DBF. Пример является только схемой. Конкретная работа с окнами рассматривается вместе с примерами программ.

```
DEFINE WINDOW kadr FROM 3,4 TO 13,50 TITLE 'КАДРЫ';
CLOSE FLOAT GROW SHADOW ZOOM
USE kadr
ACTIVATE WINDOW kadr
  <работа в окне kadr>
DEACTIVATE WINDOW kadr
  <работа с экраном>
MOVE WINDOW kadr TO 15,10
ACTIVATE WINDOW kadr
  <работа в перемещенном окне>
RELEASE WINDOWS kadr
  <работа только с экраном, окно удалено>
```

Использование окон является чрезвычайно удобным и эффективным средством создания пользовательского интерфейса в программе.

Окна очень широко используются как в оболочке FoxPro, так и в прикладных программах. Возможно использование окон некоторыми объектами (имеющими опцию WINDOW) и без предварительной активации окон. К ним относятся CHANGE/BROWSE-окна, окна редактирования текстовых данных (команды MODIFY COMMAND/FILE/MEMO), вывод доступного мемо-поля по клавишам Ctrl-Home.

Последнее закрепление осуществляется с помощью команды

- SET WINDOW OF MEMO TO <окно>

Отмена закрепления выполняется той же командой без параметра.

MESSAGE-сообщения также могут быть направлены в специальное <окно> с помощью команды

- SET MESSAGE WINDOW <окно>

Возвращение к экрану осуществляется командой, но без параметра <окно>. Например, в следующем фрагменте программы MESSAGE-сообщения выводятся в окно MES.

```
USE kadr
DEFINE WINDOW mes FROM 20,3 TO 23,60  && Описание окна
SET MESSAGE WINDOW mes                && Закрепление окна
@ 5,5 GET fam MESSAGE 'Введите фамилию имя отчество'
@ 7,5 GET tab MESSAGE 'Введите табельный номер'
ACTIVATE WINDOW mes                   && Активация окна
READ
CLEAR WINDOWS                         && Удаление окна
```

Вместе с тем окна требуют выделения заметных объемов памяти. Расходование основной памяти компьютера может быть оценено с помощью команды DISPLAY MEMORY. Так, для окна с размерами экрана требуется около 4 Кбайт памяти. Ввиду этого в больших программах следует не сразу описывать все нужные окна, а делать это по мере необходимости, уничтожая уже использованные.

В FoxPro имеется возможность не только перемещать окно, но и одновременно управлять его размером с помощью следующей команды:

■ ZOOM WINDOW <окно> MIN [AUTO]/MAX/NORM
[AT <Y1,X1>/FROM <Y1,X1> [SIZE <Y2,X2>/TO <Y3,X3>]]

Такое <окно> предварительно должно быть описано командой DEFINE WINDOW. Команда действует и на неактивное окно (до выполнения команды ACTIVATE WINDOW).

Опции команды:

MIN — устанавливает минимальный размер окна (виден только заголовок). Это позволяет разместить на экране/окне множество окон. Окно должно быть ранее описано с опцией MINIMIZE.

MAX — "распахивает" окно во весь экран/окно. Окно должно быть описано с опцией ZOOM.

NORM — окно принимает нормальный размер, заданный в описании окна или установленный ранее другой командой ZOOM WINDOW.

AUTO — опция работает только со словом MIN. При этом минимизируемое окно автоматически размещается в правом нижнем углу экрана/окна.

AT <Y1,X1>/FROM <Y1,X1> — устанавливает новые координаты левого верхнего угла минимального и нормального окна.

SIZE <Y2,X2>/TO <Y3,X4> — устанавливает новый "нормальный" размер окна в Y2 строк и X2 колонок или указывает (Y3,X4) координаты правого нижнего угла окна на экране/окне.

Пр и м е р. Приводятся команды трансформации окна OKNO.

```
.DEFINE WINDOW okno FROM 5,4 TO 10,8 MINIMIZE ZOOM
.ACTIVATE WINDOW okno
.ZOOM WINDOW okno MAX
.ZOOM WINDOW okno MIN AUTO
.ZOOM WINDOW okno NORM
.ZOOM WINDOW okno NORM AT 10,20 TO 18,40
```

Следующий пример является программой эффектного медленного "распахивания" окна OKNO1, заполненного символом *, от середины экрана к его границам. Здесь использованы четыре переменные I,J,I1,J1 для движения от центра вверх, влево, вниз и вправо соответственно.

```
DEFINE WINDOW okno1 FROM 10,39 TO 12,41 FILL '*'
ACTIVATE WINDOW okno1
```



```

STORE 12 TO i,i1
STORE 40 TO j,j1
DO WHILE i#0.OR.j#0.OR.i1#24.OR.j1#79
  i=IIF(i>0,i-1,0)
  j=IIF(j>0,j-2,0)
  i1=IIF(i1<24,i1+1,24)
  j1=IIF(j1<79,j1+2,79)
  ZOOM WINDOW okno1 NORM AT i,j TO i1,j1
ENDDO

```

В заключение рассмотрим еще одну команду, которая хотя и не является "оконной", но реализует действие, похожее на перемещение окна. Эту команду, например, удобно использовать для быстрого изменения положения какой-то области экрана.

■ SCROLL <Y1,X1>,<Y2,X2>,<вырN1> [,<вырN2>]

Команда перемещает прямоугольную область экрана с указанными координатами левого верхнего и правого нижнего углов <Y1,X1>,<Y2,X2> на <вырN1>/<вырN2> число строк/столбцов вверх/вправо. Выражения могут быть и отрицательными. Тогда перемещение будет совершаться вниз/влево. Освобождаемая область экрана очищается.

Пример:

```
SCROLL 5,10, 15, 20, 3, -6
```

Область с координатами 5,10 на 15,20 переместится вверх на три строки и влево на шесть колонок.

Для работы одновременно с несколькими окнами используется ряд понятий: Существующее Окно — окно, определенное командой DEFINE WINDOW.

Активное Окно — Существующее Окно, вызванное командой ACTIVATE WINDOW.

Окно Выдачи — Активное Окно, в которое идет выдача данных. Такое окно активировано самой последней командой ACTIVATE WINDOW.

Окно Переднего Плана — Активное Окно, которое с помощью мыши, клавиш Ctrl-F1 или команды SHOW WINDOW сделано самым верхним. В этом окне появляется курсор. Выдача, однако, все равно производится в Окно Выдачи.

При работе с единственным окном оно отвечает всем перечисленным определениям одновременно.

Оконные функции

Поскольку пользователь обычно может произвольно перемещаться по всем открытым окнам, чтобы правильно реагировать на его действия, программа должна иметь возможность распознавать имя текущего/нового окна, его статус и размер. Следующие функции позволяют выяснить эту информацию.

- WEXIST(<окно>) — возвращает .T., если заданное <окно> существует (определено командой DEFINE WINDOW), и .F. в противном случае.
- WONTOP([<окно>]) — возвращает имя Окна Переднего Плана, если такого нет, возвращается пустая строка. При включении аргумента <окно> выдается .T., если указанное окно является окном переднего плана, и .F. — если нет.
- WOUTPUT([<окно>]) — функция работает совершенно аналогично предыдущей, но в отношении определения Выходного Окна.
- WMINIMUM([<окно>]) — возвращает значение .T., если текущее или

указанное <окно> находится в минимальном состоянии.

- **WMAXIMUM**([<окно>]) — то же, но в максимальном состоянии.
- **WCOLS**([<окно>]) — число колонок в текущем или указанном <окне>.
- **WROWS**([<окно>]) — то же, но число строк <окна>.
- **WCOL**([<окно>]) — номер левой колонки текущего или указанного <окна> относительно экрана.
- **WLROW**([<окно>]) — возвращает номер верхней строки текущего или указанного <окна> относительно экрана.
- **WLAST**([<окно>]) — возвращает .Т., если указанное <окно> было активно перед текущим окном, и .F. в противном случае. Если имя окна не указано, возвращается имя последнего активного окна перед текущим окном.
- **WTITLE**([<окно>]) — возвращает заголовок (TITLE) текущего или указанного окна.
- **SCOLS**() — возвращает число колонок экрана.
- **SROWS**() — то же, но число строк экрана.

Последние две функции могут понадобиться в случае переключения экрана в режим плотной выдачи командой SET DISPLAY.

Замечание к назначению имен для BROWSE-окон. BROWSE-окно, имеющее опцию TITLE или помещенное в окно с этой опцией, далее в качестве имени использует этот заголовок, а не имя, определенное в команде DEFINE WINDOW. Причем заголовком считаются только латинские буквы и цифры (начальный символ — обязательно буква) до первого пробела, специального знака или, например, русской буквы. Таким образом, если в заголовке использованы буквы кириллицы или другие знаки, команды закрытия/активации окон их в качестве заголовка не распознают. Если закрытие таких окон выполняется непосредственно пользователем (клавишами Escape и Ctrl-W/End), то это несущественно. Однако, если нужно закрыть или активировать BROWSE-окна программным образом, данное обстоятельство следует учитывать.

Здесь можно поступить двумя способами. Во-первых, совсем не использовать опцию TITLE. В этом случае заголовком BROWSE-окна является имя самой базы данных. Чтобы это имя не было видно в рамке окна, его можно выкрасить в цвет фона, но тогда не будет видно и символов управления окном. Во-вторых, в начале заголовков всех BROWSE-команд поместить какие-то латинские символы. Например, пронумеровать окна латинскими буквами (I, II, III и т.д.), либо указать имена используемых функциональных клавиш (F1, F2 и т.д.) либо в качестве первых букв русских слов использовать латинские символы (если их начертания совпадают).

Эта проблема естественным образом решена в русифицированном пакете FoxPro.

Глава 21. МОДУЛЬНОСТЬ ПРОГРАММ

В FoxPro предусмотрена широкая возможность использования процедур, которые могут быть как внешними (в виде отдельных программных файлов), так и внутренними (внутри программы). Частным случаем процедуры является процедура-функция.

Эти средства удобно использовать для реализации некоторых одинаковых процессов обработки данных. Тогда такой процесс программируется один раз в виде одного из перечисленных выше средств и вызывается в системе по мере необходимости.

Кроме того, модульное программирование позволяет разработчику гораздо лучше ориентироваться в своей системе, а также сделать более эффективным процесс ее написания и отладки.

Встречая обращение к процедуре, FoxPro ищет ее в следующей последовательности:

1. В текущей процедуре;
2. В процедурном файле, если он подключен;
3. Снизу вверх в старших процедурах относительно текущей, если они есть;
4. На диске в виде отдельной программы.

Рассмотрим средства структурирования.

Внешние процедуры

Внешняя процедура — это совокупность команд, осуществляющих обычно какие-то законченные действия по обработке данных и образующих отдельный командный файл, т.е. отдельную программу (модуль).

Обращение к процедуре выполняется командой DO:

```
■ DO <имя командного файла>  
[WITH <список параметров>] [IN <файл>]
```

Если не указано расширение имени файла, процедура ищется СУБД на диске в следующей последовательности: сначала в EXE-, затем в APP-, FXP-и, наконец, в PRG-файле. Процедура вообще может находиться внутри другого IN <файла> как его внутренняя процедура.

В процедуру могут быть переданы и из нее получены некоторые величины, указанные в списке параметров после слова WITH. Этими величинами могут быть не только переменные и константы, но любые разрешенные выражения. В таком случае первой командой в вызываемой процедуре должна быть команда, воспринимающая их:

```
■ PARAMETERS <список параметров>
```

Параметры, указанные в команде DO после слова WITH в вызываемой программе, как принято в алгоритмических языках, называются фактическими, а соответствующие им параметры в вызываемой процедуре (в команде PARAMETERS) — формальными. Программист должен заботиться о том, чтобы передаваемые параметры по типу соответствовали друг другу. Число формальных и фактических параметров может и не совпадать. Если формальных параметров больше, избыточные параметры получают значения .F., если меньше, следует сообщение об ошибке.

Распознавать число фактически переданных параметров можно с помощью функции

```
■ PARAMETERS()
```


Завершаются файлы-процедуры одним из следующих образов:

1. Достижением последней команды файла — осуществляется возврат в старшую вызвавшую программу или на командный уровень, если это самый старший модуль;

2. Командой RETURN — возврат в старшую программу;

3. Командой CANCEL — выход на командный уровень;

4. Командой QUIT — выход из FoxPro в DOS.

Команда RETURN может содержать опции

■ RETURN [TO MASTER/<Процедура>/<выр>]

Здесь фраза RETURN TO MASTER указывает на переход к самому верхнему уровню вызывающих процедур, а RETURN TO <Процедура> — возврат на процедуру с указанным именем. Элемент <выр> применяется для передачи в вызывающую программу результата, если процедура используется как процедура-функция.

В первом и втором случаях возврат в старшую программу выполняется на команду, следующую за той, от которой произошел вызов подпрограммы. Однако при некоторых обстоятельствах бывает желательно вернуться именно на команду вызова. Это можно сделать командой

■ RETRY

Подчиненность модулей в общем довольно условна. В FoxPro допускаются даже рекурсии, т.е. обращение процедуры к самой себе.

Внутренние процедуры

Использование внешних процедур означает необходимость загрузки в память компьютера соответствующего командного модуля всякий раз, когда он вызывается командой DO. Очевидно, что это неэффективно для часто повторяющихся действий, и в этом случае лучше воспользоваться внутренними процедурами.

Совокупность команд, составляющих внутреннюю процедуру, должна начинаться командой

■ PROCEDURE <имя процедуры>

Для обозначения конца процедуры с возвратом в вызывающий модуль может использоваться команда RETURN, хотя она и не обязательна.

Процедурный файл

Множество внешних, обычно "родственных" процедур иногда удобно объединить в один процедурный файл (также типа PRG), который вызывается (загружается в память) командой

■ SET PROCEDURE TO <имя процедурного файла>

Только после этого можно командой DO вызывать из него отдельные процедуры.

В каждый момент может быть открыт только один процедурный файл. Чтобы его закрыть, нужно использовать команду SET PROCEDURE TO без параметров.

Пример. Ниже приведена структура некоторой программной системы, состоящей из главного модуля GLAV.PRG и двух процедур: внешней процедуры PROCVN.PRG и процедурного файла PROC.PRG, который, в свою очередь, содержит две процедуры PROC1 и PROC2. Главный модуль включает также внутреннюю процедуру PROCVT.

Обращение к процедурам происходит непосредственно командами DO PROCVT и DO PROCVN.

С процедурой PROCVN связь осуществляется с передачей параметров. Фактические параметры 25, A*B, 'СУММА', X передаются в процедуру, где они замещают формальные C, D, E, L, обрабатываются там и возвращаются в главный модуль. То есть здесь имеется следующее соответствие между параметрами:

25 = C, A*B = D, 'СУММА' = E, X = L.

Хотя все без исключения фактические параметры передаются в подпрограмму, обратно здесь возвращается только L в X, так как остальные фактические параметры являются либо константами (25, 'СУММА'), либо функцией (A*B).

Кроме того, здесь присутствует один процедурный модуль PROC.PRG с двумя процедурами PROC1 и PROC2. Процедурный файл загружается командой SET PROCEDURE TO PROC, и тогда из него могут выбираться сами процедуры (здесь PROC1 и PROC2). После того как необходимость в них отпадает, файл PROC.PRG удаляется из памяти командой SET PROCEDURE TO. Затем при необходимости аналогичным образом может быть загружен другой процедурный модуль.

```

*-----Главный командный модуль GLAV.PRG-----
DO procvn WITH 25,a*d,'СУММА',x      && Вызов внешней процедуры PROCVN
DO procvt                             && Вызов внутренней процедуры PROCVT
SET PROCEDURE TO PROC                && Загрузка процедурного файла PROC
DO proc1                             && Вызов процедуры PROC1
DO proc2                             && Вызов процедуры PROC2
SET PROCEDURE TO                     && Закрытие процедурного файла
PROCEDURE procvt                     && Внутренняя процедура PROCVT
RETURN                               && Конец процедуры PROCVT
*-----Конец главного модуля-----

*----Модуль PROCVN.PRG-----
PARAMETERS c,d,e,l
RETURN &&--Конец модуля---

*----Модуль PROC.PRG-----
PROCEDURE proc1
RETURN
PROCEDURE proc2
RETURN &&--Конец модуля--

```

Начало и конец каждого модуля, а также все вызовы указаны комментариями, начинающимися со знаков * или &&.

Процедуры-функции

Часто встречающиеся процессы, результатом которых является значение единственной переменной, удобно оформить в виде процедуры-функции, т. е. функции, определяемой пользователем (пользовательской функции — ПФ).

Технически ПФ может быть реализована как внешняя, так и внутренняя процедура, но доступ к ней более простой — только по имени, которое может

непосредственно включаться в команды, например, в команды: STORE, REPLACE, IF, WHILE, CASE, DISPLAY, ?, LIST, @...SAY...GET...VALID и др.

После имени ПФ должны стоять обязательные для всех функций в FoxPro скобки — пустые или с аргументами.

Внутренняя ПФ должна начинаться с команды

■ FUNCTION <Имя функции>

Если в ПФ передаются параметры, второй командой обязательно должна быть команда PARAMETERS.

Завершается ПФ командой RETURN вида RETURN <выр> где <выр> является результатом функции. Если эта команда опущена, подразумевается RETURN .T. .

FoxPro не делает различий между процедурами и процедурами-функциями. К ПФ можно обращаться и как к процедуре — командой DO. В этом случае при возврате в исходную программу значение <выр> в команде RETURN игнорируется.

Пр и м е р. Положим, требуется создать ПФ, выясняющую максимальное значение из двух произвольных величин X и Y. Назовем ПФ именем MAXIMUM. Очевидно, она должна иметь два аргумента:

MAXIMUM(x,y).

Такая функция может быть реализована в виде внешней (см. ниже слева) или внутренней процедуры с именем MAXIMUM (справа) или входить в некоторый процедурный файл, который до первого к ней обращения должен быть открыт командой SET PROCEDURE.

```
*---Модуль MAXIMUM.PRG---  
PARAMETERS a,b  
RETURN IIF(a>b,a,b)
```

```
*---Процедура-функция MAXIMUM  
FUNCTION maximum  
PARAMETERS a,b  
RETURN IIF(a>b,a,b)
```

Формальные переменные в обоих случаях — это переменные A,B. Командой RETURN "наружу" сразу передается результат, который замещает имя функции.

ПФ могут замещать практически любые <выражения>, присутствующие в командах/функциях FoxPro. При этом важно, чтобы команда RETURN возвращала из ПФ значение допустимого в данном случае типа (символьного, числового, логического, даты). Если возвращать ничего не нужно, должно быть указано хотя бы пустое (для данного типа) значение. Так, если в команде, ссылающейся на функцию, задано <вырC>, команда RETURN должна вернуть текстовое значение или хотябы нулевую строку (RETURN "").

Ниже приведен пример программы, в которой используются разные реализации ПФ. Здесь предъявляются на редактирование три поля из базы KADR.DBF. ПФ используются для сопровождения команд @...GET подсказками. Такие подсказки (MESSAGE-сообщения) организуются с помощью опций MESSAGE <вырC>. Для поля FAM сообщение о необходимости ввода фамилии прописными буквами прямо включено в команду @...GET как константа. Для поля POL подсказка реализуется в функции PL(). Для поля PODR (подразделения) подсказка имеет более сложную структуру — это перечень всех подразделений предприятия, выведенный в специальное окно PD с помощью команды LIST. Поскольку

функция никаких параметров не возвращает, после слова RETURN стоят только два апострофа. Это обеспечивает передачу назад переменной строкового типа. Если опустить этот аргумент или все слово RETURN, будет возвращено по умолчанию RETURN .1., что вызовет ошибку и прерывание. В целях деактивации окна PD при покидании поля PODR в опции VALID используется ПФ DEAC(). Так как аргументом опции является выражение логического типа (<вырL>), то команду RETURN можно опустить, поскольку RETURN .1. нас вполне устраивает.

```

SET TALK OFF
USE kadr IN a                                && Открытие баз данных
USE podr IN b
SET HEADING OFF                             && Подавление заголовка в команде LIST
DEFINE WINDOW pd FROM 1,50 TO 23,68         && Окно для команды LIST
CLEAR
@ 1,2 SAY 'Фамилия' GET fam MESSAGE 'Заглавными буквами'
@ 3,2 SAY 'Пол' GET pol MESSAGE pl()
@ 5,2 SAY 'Подразделение' GET podr MESSAGE pd() VALID deac()
READ CYCLE

FUNCTION pl                                &&----- Функция подсказки для поля POL
RETURN 'Введите М или Ж'                 && Возврат результата

FUNCTION pd                                &&----- Функция подсказки для поля PODR
ACTIVATE WINDOW pd                       && Активация окна PD
CLEAR
SELECT b                                 && Переход в область В
LIST b.podr OFF                          && Вывод всех подразделений из базы PODR.DBF
SELECT a                                 && Возвращение в область А
RETURN ''                                && Возврат пустого слова

FUNCTION deac                             &&----- Функция деактивации окна
DEACTIVATE WINDOW pd

```

Параметры, передаваемые в функцию через команду PARAMETERS, хотя и могут изменяться внутри функции, в вызывающей программе останутся неизменными, поскольку они передаются "по значению". Способом передачи параметров управляет команда

■ SET UDFPARMS TO VALUE/REFERENCE

По умолчанию VALUE. Если мы хотим передать назад не только основной результат через команду RETURN, но и какие-то из измененных параметров, следует воспользоваться установкой REFERENCE ("по ссылке").

В процедуры и программы параметры всегда передаются "по ссылке", если только они не взяты в скобки. Тогда — "по значению".

Область действия переменных

Переменные могут передаваться в программы и процедуры не только списком фактических параметров, которые заменяются в них на формальные, но и непосредственно. Переменные, введенные в старшем модуле, существуют также во всех подчиненных модулях и могут быть в них обработаны нужным образом, т. е. специально параметры можно не передавать.

Рассмотрим статус и область действия переменных.

Глобальные переменные. Переменные и массивы, будучи объявленными как глобальные, имеют силу всюду в среде СУБД — во всех ее программах, подпрограммах и процедурах. Более того, они остаются в СУБД, даже если создавшая их программа закончена и удалена из памяти. Такие переменные объявляются командой

■ PUBLIC <переменные> [, <массивы переменных>]

Например, команда

```
PUBLIC A, B, R(10), Z(2,6)
```

объявляет глобальными переменные A и B и массивы R и Z. Применение команды DIMENSION для объявления этих массивов больше не требуется. Подобный же эффект имеет использование имен переменных в командном окне. После выполнения, например, команд

```
.A=3  
.B='КНИГА'
```

переменные A и B будут считаться глобальными (PUBLIC) и останутся в памяти в случае загрузки в нее программ, где могут быть изменены.

Этот эффект удобен, например, для отдельной отладки командных модулей. Предположим, что некоторый отлаживаемый модуль нуждается в данных, вырабатываемых другим модулем, который еще не существует или имеется, но мы не хотим тратить каждый раз время на его загрузку и исполнение. В этом случае лучше задать исходные данные вручную в командном окне, и они остаются в памяти. Затем вызываем отлаживаемый модуль на исполнение, выявляем ошибки, запускаем его снова и т.д. При этом исходные данные для него всегда доступны.

Локальные переменные. Переменные, объявляемые локальными (PRIVATE), существуют только в данном модуле и всех младших по отношению к нему модулях в моменты их вызова.

■ PRIVATE <переменные>

Возможно объявление локальными групп переменных без их поименного перечисления.

■ PRIVATE ALL LIKE/EXCEPT <маска>

Параметры LIKE и EXCEPT разъяснены ранее в команде SAVE.

Команда PRIVATE применяется для временного (только внутри данной и подчиненных ей процедур) освобождения переменных, существующих в старших модулях или имеющих статус PUBLIC. При этом указанные переменные не уничтожаются, но делаются как бы невидимыми для данного модуля, и их имена могут быть использованы программистом по своему усмотрению. При выходе из указанного модуля все PRIVATE-переменные утрачиваются, а старые переменные восстанавливаются.

Глава 22. ПРИМЕРЫ ПРОГРАММ В СРЕДЕ СУБД FoxPro

Рассмотрим несколько типовых задач, обычно встречающихся в практике программиста:

- о поддержка базы данных;
- о древовидная организация данных;
- о работа со словарями.

22.1. Поддержка базы данных

Поддержка или ведение базы данных подразумевает наличие средств доступа к данным и средств управления ими. При этом рабочий интерфейс должен быть максимально "комфортным".

Здесь нужно решить, что пользователь будет обычно видеть на экране компьютера. Удобно в качестве "рабочего стола", на котором он будет принимать решения и совершать действия по обработке данных, взять не какое-то меню, а отображение самой базы данных. На экране должны присутствовать и указания на возможные действия пользователя. Поскольку на нем никогда не хватает места, указания на некоторые действия можно поместить в разветвляющиеся световые меню. При необходимости также должна быть предусмотрена контекстно-зависимая помощь (HELP — F1).

Рассмотрим средства ведения баз данных в следующей последовательности:

- о функции поддержки базы данных;
- о средства предъявления данных;
- о органы управления и совмещение их с окнами редактирования.

Функции поддержки базы данных. Для обслуживания базы необходимы следующие функции (в скобках указаны соответствующие команды).

1. Перемещение вперед/назад на одну запись (SKIP/SKIP -1). При этом должны блокироваться попытки выйти за первую/последнюю запись базы. Достижение верхней/нижней границы желательно отображать на экране.

2. Переход на начало/конец базы (GO TOP/BOTTOM).

3. Пометка записей, предназначенных для удаления, и снятие пометки (DELETE/RECALL). Помеченную запись желательно как-то выделить.

4. Дополнение базы новой записью (APPEND BLANK).

5. Выход из окна редактирования.

6. Поиск по ключу (SEEK, LOCATE). В случае неудачного поиска указатель записей должен вернуться на исходную запись.

7. Вывод/печать данных.

8. Упаковка данных (PACK).

9. Переиндексация базы в случае аварийного завершения предыдущего сеанса работы (REINDEX).

10. Страховое копирование всей базы данных в другую директорию или с другим именем (COPY).

11. Полная очистка базы от данных (ZAP).

Из вышеперечисленных функций пункты 1-5 являются самыми употребимыми. Такие операции должны осуществляться пользователем в "одно касание" — одним нажатием клавиши или кнопки мыши. За этими действиями лучше закрепить привычные клавиши, что в некоторых случаях не потребует их отображения на экране. Обычно клавиши PgUp и PgDn являются средством "листания" базы. Клавиши Ctrl-PgUp и Ctrl-PgDn удобно использовать для перехода в начало/конец базы. При этом придется пожертвовать этими клавишами как входами в мемо-поля, но останется еще Ctrl-Home. Конечно, если вы используете команду @ ... EDIT, эти клавиши не нужны.

Пометка к удалению записи может выполняться любой клавишей, однако

удобно воспользоваться, например, Ctrl-T (как в команде BROWSE). На случай ошибочной пометки должно быть предусмотрено и снятие пометки, которое обычно осуществляется той же клавишей. Здесь следует иметь в виду, что, как правило, при работе с базой данных действует установка SET DELETED ON: из дальнейшей обработки изымаются все помеченные записи. Следствием этого является то, что, если такая запись покинула экран, больше увидеть ее будет нельзя.

Дополнение базы новой записью можно реализовать с помощью любой клавиши, но удобно воспользоваться уже привычной в случае команды BROWSE клавишей Ctrl-N.

Для выхода из режима редактирования обычно не требуется закреплять какие-то специальные средства. Здесь следует использовать возможности клавиш Escape и Ctrl-W/End. Однако вполне можно представить себе ситуацию, когда существует несколько "выходов".

Функции поиска и вывода данных могут быть реализованы как доступные сразу, так и как скрытые, в зависимости от того, как часто они нужны.

Необратимые, потенциально опасные или редко используемые действия не должны быть постоянно доступными пользователю и информация о них не должна загромождать экран. Так, упаковка и переиндексация, хотя и не опасные, но редкие и медленные действия, которые обычно выполняются сразу для всех баз, обрабатываемых в прикладной системе. Эти функции, как правило, размещаются в центральном меню.

Полная очистка базы является чрезвычайно редким и ответственным шагом, и доступ к этой функции должен быть умышленно затруднен для избежания какой-либо случайности.

Команды предъявления данных. Теперь обсудим средства редактирования данных в базе данных.

Данные из базы можно предъявлять различным образом в зависимости от используемых команд.

1. Команды BROWSE или CHANGE:

BROWSE/CHANGE <заголовки, шаблоны, вызовы функций>

В этом случае автоматически реализуются следующие функции: удаление (Ctrl-T) и дополнение (Ctrl-N) записей, перемещение между записями, листание экранов, выход. Остальные перечисленные в постановке задачи функции должны быть реализованы программистом.

2. Команды CHANGE/READ с форматным файлом:

SET FORMAT TO <форматный файл>
CHANGE или READ
SET FORMAT TO

Эта форма предъявления данных позволяет отойти от стандартного облика CHANGE-окна, где поля могут быть расположены только вертикально и имеются лишь очень скудные возможности для указания каких-либо подсказок непосредственно в окне (хотя они могут быть вызваны с помощью процедур, например нажатием клавиши F1, или в другом окне/экране). Теперь поля можно располагать как угодно, использовать рамки и цвета, давать указания и предупреждения пользователю по работе в окне. Однако одновременно здесь утрачивается возможность удаления и дополнения данных и, кроме того, при достижении границ базы или границ действия команды происходит несанкционированный выход из команд.

3. Команда READ с командами @...SAY...GET:

@...SAY...GET...
@...SAY...GET...
READ

Эта форма предъявления данных является самой гибкой и дает непосредственный доступ к полям базы. Теперь кроме собственно редактирования, поля базы данных можно анализировать, трансформировать, запоминать в переменных или в других базах и т.д.. Но никаких встроенных функций, помимо, естественно, редактирования, такая команда READ не реализует. О них нужно позаботиться самим.

По аналогии с CHANGE/BROWSE-окном такой режим будем иногда называть READ-окном, хотя использование команд @ ... GET, READ не обязывает помещать их в специальное окно — они могут проектироваться и непосредственно на экран.

Все операции над базой, предъявленной командой READ (например, операции перемещения), могут быть совершены либо после завершения команды READ, либо в вызываемых процедурах.

В первом случае в программе необходимо предусмотреть средства возврата к редактированию. Для этого команда включается в цикл вида DO WHILE .t. ... ENDDO, который после завершения какого-то действия регенерирует окно редактирования, созданное командой READ. Внутри цикла в структуре DO CASE...ENDCASE с помощью функции LASTKEY() или READKEY() могут анализироваться и обрабатываться только клавиши, приводящие к выходу из команды READ, после чего происходит возврат (ENDDO) к окну редактирования. Это клавиши PgUp, PgDn, Ctrl-Home, Ctrl-PgUp, Ctrl-PgDn, Ctrl-End, Ctrl-W, Escape. Однако следует учитывать, что клавиши Ctrl-Home, Ctrl-PgUp, Ctrl-PgDn являются средством доступа и к мемо-полям (если они имеются в базе). То есть их нажатие, когда курсор находится в мемо-поле, вызовет открытие окна для его редактирования. Чтобы этого избежать, курсор должен находиться в другом поле.

Рассмотренный механизм (см. фрагмент ниже слева) широко используется в предыдущих версиях FoxPro, а также в СУБД dBASE и Clipper. Его недостатками являются необходимость поддержания цикла DO WHILE и незначительное число клавиш выхода из команды READ, которые можно задействовать для собственных нужд.

Во втором случае процедуры могут быть вызваны как командами вида ON KEY LABEL, так и средствами управления в стиле Windows (кнопками). После завершения такой процедуры происходит автоматический возврат к исходному экрану. Поскольку команда READ при этом каждый раз непосредственно не используется, сам экран (кроме поля на котором находится курсор) не обновляется. Ввиду этого необходимо в процедурах предусмотреть команду обновления экрана SHOW GETS. Если выход из редактирования должен быть осуществлен в процедуре, она должна включать команду CLEAR GETS.

При таком подходе удобно использование опции CYCLE в команде READ, которая блокирует случайный выход из экрана редактирования. При этом естественный выход из команды READ теперь возможен только с помощью клавиш Escape и Ctrl-W/End (см. ниже справа).

DO WHILE .t.	@... SAY... GET...
@... SAY... GET...	@... SAY... GET...
READ	READ CYCLE
DO CASE	PROCEDURE <...>
CASE LASTKEY()=...	SHOW GETS && обновление
<анализ нажатий и>	и возврат
<обработка данных>	
CASE LASTKEY()=...	PROCEDURE <...>
ENDCASE	CLEAR GETS && выход
ENDDO	

Именно такую технологию, как более удобную, мы и будем использовать в дальнейшем.

Как видим, по мере отхода от стандартных форм отображения данных в базе данных утрачивается все больше "стандартных" функций, но одновременно программисту предоставляется все больше возможностей для управления процессом обработки данных. Кроме того, помимо типичных функций все равно есть такие, которые нужно реализовывать программно.

Все перечисленные формы отображения данных могут иметь место в прикладных системах обработки данных, тем более, что и стандартные окна редактирования в FoxPro обладают исключительно широкими возможностями. Обычно для отображения "узких", т.е. имеющих небольшое число полей, баз достаточно просто одной команды BROWSE. Если база значительно шире горизонтального размера экрана или при вводе предполагается сложный анализ/трансформация данных, приходится обращаться к команде READ. Иногда бывает удобно совместить обе формы предъявления данных. С помощью команды BROWSE тогда создается как бы видимое "оглавление" базы данных, через которое пользователь может легко ориентироваться и перемещаться внутри нее. Доступ к данным в этом режиме ограничен или вообще заблокирован. При необходимости обратиться к данным на фон, созданный командой BROWSE (по специально закрепленной клавише), проектируется окно ввода READ только для одной текущей записи. Такие окна могут работать и "параллельно", как показано в одном из примеров.

Средства управления. Необходимо также выбрать вид меню, сопровождающего экран редактирования данных. FoxPro позволяет любой тип меню (световое, клавишное, кнопочное) совместить с любой командой редактирования (CHANGE/BROWSE/READ) данных, причем меню может быть как постоянно доступным, так и вызываемым.

Выбор соответствующего меню может быть сделан в зависимости от желаемой скорости доступа к нему, а также возможности работы с клавиатурой и мышью. Хотя применение мыши обычно очень удобно, бывает, что пользователь не хочет или не может (например, ввиду ее отсутствия или стесненности рабочего места) применять мышь. Вместе с тем программные продукты, созданные не под конкретного заказчика, по возможности должны предусматривать удобную работу как только с клавиатурой, так и с мышью.

Клавишное меню реализуется с помощью команды вида ON KEY LABEL (иногда команд SET FUNCTION) очень быстрый вызов любой закрепленной процедуры с возвратом в окно редактирования.

Для того чтобы клавишные назначения действовали только в нужных местах программы, при входе в вызываемую процедуру они должны отключаться (ON KEY), а при ее завершении — восстанавливаться (ON KEY LABEL...). Этот процесс удобнее осуществлять с помощью стека клавишных назначений (команды PUSH KEY, POP KEY). Здесь следует обратить внимание на то, чтобы выход из процедур всегда шел в том же порядке, что и их вызов. Если оператор имеет возможность изменить этот порядок, стеком следует пользоваться осторожно, сочетая его с командами ON KEY.

Процедуры клавишного меню, в свою очередь, могут вызывать световые меню. Само клавишное меню является просто строчкой на экране и нечувствительно к мышью. Вместе с тем, поскольку в языке FoxPro имеется возможность зафиксировать текущее местоположение маркера мыши, можно назначить на каждый элемент клавишного меню вызов тех же процедур и от кнопки мыши.

Световое меню в зависимости от способа подключения и типа окна редактирования может работать по-разному. Если световое меню комбинируется с BROWSE/CHANGE-окном, то оно должно активироваться

командой `ACTIVATE MENU` с опцией `NOWAIT` до команды `BROWSE`. При этом курсор вначале находится в `BROWSE`-окне, но может быть переброшен мышью в меню и обратно. Выход из меню, конечно, возможен и при нажатии клавиши `Escape`. Недостатком такого использования светового меню совместно с `BROWSE`-окном является то, что изменения данных, сделанные через меню, отображаются в окне редактирования только при возврате в него курсора.

Это не относится к `BAR`-меню, если оно определяется командой `DEFINE BAR` с опцией `BAR` (т.е. в стиле системного меню). Здесь изменение данных сразу отражается в `BROWSE`-окне, поскольку после выбора определенного пункта меню курсор автоматически возвращается назад.

За всем `BAR`-меню и любыми его `PAD`-пунктами, как известно, можно закрепить "горячие" клавиши с помощью опций `KEY` в командах `DEFINE`. В этом случае переход к `BAR`-меню становится возможным не только с помощью мыши, но и этих клавиш. Возврат по-прежнему должен осуществляться нажатием клавиши `Escape`.

Если перед командой `BROWSE` задействовать не команду `ACTIVATE MENU...NOWAIT`, а команду `SHOW MENU`, то применять мышь больше будет нельзя, но зато все меню и/или любые его `PAD`-пункты могут быть вызваны нажатием "горячих" клавиш с последующим возвратом назад.

Совмещение светового меню с вводом по команде `READ` может быть осуществлено точно так же. Однако использование команды обновления `GET-областей` ввода `SHOW GETS` в конце всех процедур, изменяющих данные, снимает все проблемы по адекватному отображению данных из базы независимо от текущего положения курсора (в меню или окне). Пример совмещения команды `READ` и светового меню будет рассмотрен в конце данного раздела.

Кнопочное меню также может быть совмещено с командой `BROWSE`, причем последняя должна использовать опцию `NOWAIT`. Обновление содержимого `BROWSE`-окна осуществляется командой `SHOW WINDOW REFRESH`.

Кнопочное управление легко сочетается с вводом по команде `READ`. Вообще `GET`-органы управления и `GET`-области ввода удобно активировать одной и той же командой `READ`. Соответствующий пример приведен в гл.26.

Программа KADR1.PRG. Основное окно с меню поиска и меню подразделений изображено на рис 22.1 (курсор стоит на самой первой записи). В его заголовке отображены функции рабочих клавиш (клавишное меню):

- F3 — поиск данных;
- Ctrl-T — дополнение базы;
- Ctrl-N — пометка записи, предназначенной для удаления;
- Ctrl-P — печать текущей записи;
- Ctrl-PgUp — переход к началу базы;
- Ctrl-PgDn — переход к концу базы;
- Ctrl-W/Q — выход из окна и завершение работы.

Кроме того, в заголовках соответствующих полей показаны клавиши входа: F7 — вызов меню подразделений для ввода подразделений и Ctrl-Home — для входа в мемо-поле `PER`.

Конечно, совершенно необязательно выносить в заголовок окна такое число сообщений. Если для них не хватает места, можно поместить их в нижнюю часть экрана, уменьшив размер `BROWSE`-окна, и/или поместить в специальное окно `HELP`, вызываемое при нажатии клавиши `F1`.

Ниже будет приведена программа `KADR1.PRG`. Рассмотрим ее.

Установка рабочей среды системы. Здесь кроме очевидных уже действий подавляющие следующие, ненужные в готовых программах средства — прерывание программы при нажатии клавиши `Escape` (`SET ESCAPE OFF`),

доступ к Help (SET HELP OFF). Очищаются все системные закрепления за F-клавишами (CLEAR MACROS). Эти команды следует внести в программу в последний момент, поскольку перечисленные средства могут понадобиться при отладке. Кроме того, по команде SET NEAR ON устанавливается положение указателя записей при неудачном поиске.

F3-поиск ^N-доп. ^T-удал. ^P-печать ... ^PgDn-конец ^W/Q-выход					
Фамилия	Родился	Табель	Пол	Подраз. - F7	Перем. - ^Home
КУЛАКОВА М.И.	15.04.49	6	Ж	ОГМ	memo
МИРОНОВ Р.И.	Выберите критерий поиска По фамилии По табелю По средней зарплате По подразделению Отмена сортировки				memo
ПОПОВ А.А.					memo
ПОТАПОВ Д.П.					memo
РОМАНОВА М.С.					memo
СИДОРОВ П.С.					memo
ЯКОВЛЕВА А.И.				ОГМ ОК КБ Бухгалтерия	memo
КУЛАКОВА М.И. Табель - 6 Подразделение - ОГМ					

Рис.22.1

Открытие баз данных. В программе предполагается ускоренный поиск по фамилии, табельному номеру, средней зарплате и подразделению. Ввиду этого в области А открывается база KADR.DBF совместно с индексными файлами KADRFAM.IDX (индекс по полю FAM), KADRTAB.IDX (индекс по полю TAB), KADRSZAR.IDX (индекс по полю SZAR), KADRPODR.IDX (индекс по PODR). Порядок предъявления записей — пока естественный (ORDER 0).

Кроме основной базы будем использовать в области В вспомогательную базу PODR.DBF, в которой хранятся названия всех подразделений. Эта база состоит из единственного символьного поля длиной 15, также названного PODR. Ее содержимое используется в меню для заполнения поля PODR в основной базе KADR.DBF, что позволит ускорить и сделать более надежным ввод информации о подразделениях, в которых работают сотрудники. Считаем этот файл существующим. Ведение такой небольшой базы не вызовет никаких трудностей — для этого достаточно команды BROWSE.

Определение окон. Определяются окна для отображения базы (окно KADR), задания ключа поиска (окно POISK) и окно PER для предъявления мемо-поля PER. Кроме того, создается окно F1, в которое будет помещен текст подсказки, вызываемой при нажатии клавиши F1.

Определение меню. Описываются два POPUP-меню. Первое (POISK) — для задания критерия поиска. В этом меню перечисляются все доступные виды поиска, а также возможность возвращения к естественному порядку отображения записей (пункт "Отмена сортировки"). Обрабатывается выбор из меню в одноименной процедуре POISK. Меню вызывается нажатием "горячей" клавиши F3 (опция KEY F3 в команде DEFINE POPUP). Второе POPUP-меню PODR предназначено для предъявления перечня подразделений. Для обработки выбора здесь не назначается какая-либо процедура, поскольку это меню требуется в двух местах программы — при вводе в базу подразделений и при задании ключа поиска. Выдача меню сопровождается подсказкой "Укажите подразделение".

Клавишные назначения (клавишное меню). Здесь закрепляются вызовы процедур/команд за всеми выбранными клавишами. Клавише Ctrl-P назначается процедура печати (PRIN), клавише Ctrl-PgUp — команда перехода в начало базы (GO TOP), клавише Ctrl-PgDn — команда перехода в конец базы (GO BOTTOM), клавише F7 — процедура F7 вызова меню

подразделений, клавише F1 — процедура помощи F1, клавише Ctrl-W — процедура CONFIG для управления сохранением конфигурации BROWSE-окна при выходе из него.

Первое действие, фактически выполняемое в программе, — это выяснение у пользователя, желает ли он работать с экраном, определенным только синтаксисом команды BROWSE, или же хочет, чтобы окно предьявлялось в форме, которую он установил в последнем сеансе работы. Диалог организуется с помощью команды WAIT в форме следующего сообщения:

Загрузить окно в стандартном (Esc) или предыдущем виде?

Если нажимается любая клавиша, кроме клавиши Escape, по команде SET RESOURCE ON делается доступным ресурсный файл FOXUSER.DBF, из которого извлекается конфигурация окна BROWSE. В противном случае этот файл игнорируется (OFF). В самой команде BROWSE режим сохранения конфигурации поддерживается опцией LAST. Применение ресурсного файла имеет смысл с широким BROWSE-окном, когда не все поля могут быть предьявлены сразу. Для данной базы этого не требуется — это лишь пример.

Команда BROWSE. С помощью команды создается облик основного окна KADR, показанный выше. Здесь предусмотрены некоторый контроль и преобразование ввода. Фамилия (поле FAM) в пользовательской функции ZAG() конвертируется в строку, содержащую только заглавные буквы. Для поля (поле POL) допускается ввод только одной из двух букв М или Ж. Если ввод ошибочный, появится сообщение "Только М или Ж". Аналогичный контроль предусмотрен для данных о семейном положении (поле SEM). Средняя зарплата (поле SZAR) ограничена сверху числом 9000. Поскольку выбор подразделения осуществляется только через меню, редактирование поля PODR заблокировано ключом :R.

Для удобства пользователя ниже экрана редактирования выводятся значения фамилии, табельного номера и подразделения из текущей записи. Этот процесс реализуется с помощью опции WHEN вызовом пользовательской функции SOOB().

Еще ниже выводится строка-подсказка разрешенных действий пользователя в некоторых полях. Она реализуется с помощью ключа :W, через который вызывается функция HLP(). При входе курсора в очередное поле функцией анализируется имя этого поля. Если для указанного поля предусмотрена помощь, в строке 22 выводится сообщение. Предварительно строка "раскрашивается" по команде @...FILL красным по красному (R/R), что соответствует ее очистке от предыдущего сообщения с одновременным выделением цветом. Сами сообщения выводятся белым по красному.

По выходе из команды BROWSE ликвидируются все клавишные назначения, окна и меню. На этом основная часть программы завершается. Ниже идут только процедуры и функции, которые мы сейчас рассмотрим. При входе практически во все процедуры программы командой PUSH KEY CLEAR отменяются предыдущие клавишные назначения и заносятся в стек, а при выходе они восстанавливаются по команде POP KEY.

Процедура CONFIG выполняется при выходе из BROWSE-окна с помощью клавиш Ctrl-W. Здесь активируется ресурсный файл и только после этого деактивируется само окно. Причем в команде DEACTIVATE WINDOW 'F3' в качестве имени используются первые два символа, указанные в опции TITLE, поскольку в качестве имени BROWSE-окна FoxPro воспринимает только латинские буквы и цифры. Выход из окна с помощью клавиш Ctrl-Q не сохраняет конфигурацию в ресурсном файле, даже если он подключен. Выход с помощью Escape будет сохранять или не сохранять конфигурацию в зависимости от того, был или не был ранее открыт ресурсный файл.

Процедура помощи F1 вызывается нажатием клавиши F1 и демонстрирует в окне F1 сообщения-подсказки о действии "горячих" клавиш.

Процедура печати (PRIN) реагирует на клавиши Ctrl-P и сначала создает окно PRIN и выводит в него для предварительного просмотра текущую запись базы. Для удобства просмотра и печати командой SET MEMOWIDTH устанавливается длина вывода мемо-поля PER (перемещения) в 62 колонки. После этого следует меню согласия на печать. Если нажата клавиша Escape, работа процедуры завершается, если другая клавиша — выясняется состояние принтера (PRINTSTATUS()). При готовности принтер активируется (SET PRINTER ON) и производится печать. В противном случае выдается сообщение 'Принтер не готов' и процедура завершается.

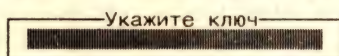
Процедура F7 активирует (ACTIVATE POPUP PODR) меню подразделений для ввода данных в поле PODR из базы KADR.DBF. Это меню проектируется на основное окно в центральной его части. Результатом выбора из меню, если не нажата клавиша Escape (LASTKEY()#27), является занесение в поле PODR базы KADR.DBF значения поля PODR из PODR.DBF.

Процедура POISK задания ключа поиска и самого поиска. Здесь в переменной Y запоминается номер отобранного пункта меню, а в переменной N — номер текущей записи. Далее в структуре DO CASE производится разбор выбора пользователя.

Если Y=5, значит, выбран пункт "Отмена сортировки" и командой SET ORDER TO 0 отключается главный индекс. Все индексы остаются открытыми, но ни один из них не является более главным и не влияет на порядок предъявления записей.

Если Y=4, главным делается индекс KADRPODR.IDX, который определит алфавитный порядок следования записей по подразделениям. Затем активируется меню подразделений и выбор пользователя заносится в переменную A. Одновременно он запоминается в переменной D, которая пригодится для включения ключа поиска в сообщение WAIT о неудачном поиске. Поскольку такое включение может быть только символьного типа, в запросах числового типа производятся необходимые преобразования функцией STR().

Все остальные (OTHERWISE) выборы из меню обрабатываются однотипным образом — активируется нужный индекс, предъявляется окно POISK



для задания содержательного ключа поиска и с помощью команд вида @ ... GET и READ вводятся искомые данные.

При вводе фамилии из нее удаляются возможные пробелы (ALLTRIM (A)), что позволяет вести поиск и по не полностью заданному ключу (фамилии), а также с помощью пользовательской функции ZAG() все буквы конвертируются в заглавные. По умолчанию (DEFAULT SPACE(25)) вначале переменная A здесь получает пустое значение длиной 25 пробелов.

Далее, если не выбрана "Отмена сортировки" (Y=5), ключ поиска не пуст (!EMPTY(a)), поиск закончился неудачно (!SEEK(a)) и это не поиск по зарплате (Y#3), выдается сообщение, например следующего вида:

Поиск По фамилии: Васин А.Н. НЕ УДАЧНЫЙ

которое затем удаляется нажатием любой клавиши, и указатель записей возвращается на (GO N) старое место. Потом происходит возврат в команду BROWSE. Если в результате поиска была найдена нужная запись, мы увидим курсор стоящим на ней.

В случае поиска по зарплате (Y=3) при отрицательном результате возврат

на старую запись не производится и сообщение не выдается. Это объясняется особенностями числовых запросов. Поскольку точную зарплату пользователю указать трудно, он может задать ее приблизительно. Если такое число не будет найдено, указатель записей остановится на записи, имеющей ближайшее большее значение (режим определяется командой SET NEAR ON). При возврате в окно редактирования пользователь может легко найти нужную запись, "пролистав" базу вблизи этого места.

Последний элемент программы — функция ZAG(). В нее передаются два параметра — имя преобразуемой переменной/поля P и его тип K. Если K=.T., значит, передано имя ПОЛЯ базы данных, если K=.F. — имя ПЕРЕМЕННОЙ. Это нужно для того, чтобы правильным образом использовать полученный результат. Если это поле, то выполняется команда замещения (REPLACE), если переменная — команда присваивания (=). Собственно конвертирование букв выполняется функцией CHRTRAN(&P,S,Z), где в поле с именем &P все символы, перечисленные в переменной S, заменяются на соответствующие символы, взятые из переменной Z. Для переменной P макроподстановка не нужна, поскольку из процедуры F передается не имя переменной, а ее значение.

Функция ZAG имеет смысл только при работе с оригинальным пакетом FoxPro и не нужна, если вы работаете с русифицированной адаптацией, где для конвертирования букв достаточно использовать символ шаблона "!" в команде @...GET. Очевидно, что это гораздо проще и преобразование происходит практически мгновенно. В дальнейшем будем считать, что эта возможность существует.

```
*-----Программа KADR1.PRG-----нужны файлы: KADR.DBF, KADRFAM.IDX,
*---KADRTAB.IDX, KADRSZAR.IDX, KADRPODR.IDX PODR.DBF и FOXUSER.DBF
SET DATE GERMAN          && Установка операционной среды
SET TALK OFF
SET DELETED ON
SET NEAR ON
SET ESCAPE OFF           && Подавление прерывания от клавиши Escape
SET HELP OFF             && Подавление вызова Help
CLEAR MACROS             && Сброс всех функциональных клавиш
ON KEY CLEAR
USE kadr INDEX kadrfam,kadrtab,kadrszar,kadrpodr ORDER 0
USE podr IN b            && Открытие базы PODR.DBF для меню подразделений
*----- Описание окон -----
DEFINE WINDOW kadr FROM 0,0 TO 18,79   && Основное окно
DEFINE WINDOW poisk FROM 8,45 TO 10,73;
  TITLE 'Укажите ключ'                && Описание окна поиска
DEFINE WINDOW per FROM 6,2 TO 10,77 TITLE 'Перемещения'
SET WINDOW OF MEMO TO per             && Окно просмотра мемо-поля PER
DEFINE WINDOW f1 FROM 1,1 TO 12,50 COLOR W+/R &&Окно помощи HELP
*----- Описание меню -----
DEFINE POPUP poisk FROM 4,14;
  TITLE 'Выберите критерий поиска' KEY F3
DEFINE BAR 1 OF poisk PROMPT 'По фамилии'
DEFINE BAR 2 OF poisk PROMPT 'По таблицю'
DEFINE BAR 3 OF poisk PROMPT 'По средней зарплате'
DEFINE BAR 4 OF poisk PROMPT 'По подразделению'
DEFINE BAR 5 OF poisk PROMPT 'Отмена сортировки' COLOR ,R/W
ON SELECTION POPUP poisk DO poisk
DEFINE POPUP podr FROM 6,36 PROMPT FIELD b.podr MESSAGE;
  'Укажите подразделение'             && Описание меню подразделений
ON SELECTION POPUP podr DEACTIVATE POPUP
*----- Назначения клавиш -----
ON KEY LABEL ctrl+p      DO prin      && Печать
ON KEY LABEL ctrl+pgup   GO TOP       && Начало базы
ON KEY LABEL ctrl+pgdn   GO BOTTOM     && Конец базы
ON KEY LABEL f7          DO f7        && Подразделения
ON KEY LABEL f1          DO f1        && Помощь
ON KEY LABEL ctrl+W      DO config    && Конфигурация
                                && Определение конфигурации BROWSE-окна
WAIT 'Загрузить окно в стандартном (Esc) или предыдущем виде?';
WINDOW
```



```

IF LASTKEY()=27                && Если нажата клавиша Escape,
    SET RESOURCE OFF          && ресурсный файл отключается
ELSE                            && В противном случае -
    SET RESOURCE ON           && подключается
ENDIF
BROWSE TITLE 'F3-поиск ^N-доп. ^T-удал. ^P-печать ^+;
    ^PgUp-нач. ^PgDn-конец ^W/^Q-выход';
FIELDS fam :H='Фамилия' :12 :V=zag('a.fam',.T.);
dtr :H='Родился' :W=hlp();
tab :H='Табель' :W=hlp();
pol :H='Пол' :V=INLIST(pol,'M','Ж') :E='Только М/Ж' :W=hlp();
det :H='Дети' :W=hlp();
sem :H='Сем.п' :V=INLIST(sem,'Б','X','P') :W=hlp();
szar :H='Ср.зар.' :B=,9000 :W=hlp();
podr :H='Подраз.-F7' :11 :R :W=hlp();
per :H='Перем.-^Home' :W=hlp();
WINDOW kadr COLOR SCHEME 10 WHEN soob() LAST
ON KEY
CLEAR WINDOW
CLEAR POPUPS
CLEAR &&-----Конец основной части программы-----

FUNCTION hlp                &&-----Функция вывода строки-подсказки
@ 22,0 FILL TO 22,79 COLOR r/r                && Очистка строки
DO CASE                && Вывод подсказки для текущего (VARREAD()) поля
CASE VARREAD()='Podr'
    @ 22,20 SAY 'Вызвать меню подразделений клавишей F7' COLOR w+/r
CASE VARREAD()='Pol'
    @ 22,27 SAY 'Введите пол - только М и Ж' COLOR w+/r
CASE VARREAD()='Sem'
    @ 22,12 SAY 'Семейное полож.: в браке (Б),'+
        ' холост (X), разведен (P)' COLOR w+/r
ENDCASE

FUNCTION soob                &&-----Функция вывода содержимого записи
@ 20,2 SAY fam COLOR W+/G                && в нижней части экрана
@ 20,30 SAY 'Табель - '
@ 20,38 SAY TAB COLOR W+/G
@ 20,47 SAY 'Подразделение - '
@ 20,63 SAY podr COLOR W+/G
RETURN

PROCEDURE config                &&---Процедура сохранения конфигурации окна
SET RESOURCE ON                && Активация ресурсного файла
DEACTIVATE WINDOW 'F3'        && Деактивация BROWSE-окна
RETURN

PROCEDURE f1                &&-----Процедура помощи HELP
PUSH KEY CLEAR                && Отключение клавиш
SET CURSOR OFF                && Отключение курсора
ACTIVATE WINDOW f1            && Активация окна
TEXT                && Предъявление текста подсказки
^N - дополнение базы новой записью
^T - пометка записи к удалению
^P - вывод/печать текущей записи
F3 - поиск записи по ключу
F7 - вызов меню подразделений
^PgUp - переход в начало базы
^PgDn - переход в конец базы
^W - выход из окна редактирования
      (конфигурация сохраняется)
^Q - то же, но конфигурация не сохраняется
^Home - вход в мемо-поле
ENDTEXT
WAIT '' NOWAIT                && Пауза для просмотра
DEACTIVATE WINDOW f1          && Деактивация окна
SET CURSOR ON                && Включение курсора
POP KEY                        && Восстановление клавиш
RETURN

PROCEDURE prin                &&-----Процедура печати
PUSH KEY CLEAR                && Отключение клавиш
DEFINE WINDOW prin FROM 4,8 TO 20,71        && Окно просмотра
ACTIVATE WINDOW prin
SET MEMOWIDTH TO 62                && Ширина выдачи мемо-поля

```



```

?' Фамилия И.О.',fam,' Табельный номер',tab
?' Пол',pol,' Дата рождения',dtr,;
?' Семейное положение',sem,' Детей',det
?' Подразделение',podr,' Средняя зарплата',szar
?' Перемещения:
?' per
WAIT 'Любая клавиша - ПЕЧАТЬ, Esc - ОТКАЗ' WINDOW NOWAIT
IF LASTKEY()#27 && Если не ОТКАЗ
  IF PRINTSTATUS() && и если принтер готов,
    SET PRINTER ON && подключение принтера
    ?' Фамилия И.О.',fam,' Табельный номер',tab
    ?' Пол',pol,' Дата рождения',dtr,;
    ?' Семейное положение',sem,' Детей',det
    ?' Подразделение',podr,' Средняя зарплата',szar
    ?' Перемещения:
    ?' per
    ?
    SET PRINTER OFF
  ELSE && Если нет - сообщение
    WAIT 'Принтер не готов' WINDOW NOWAIT
  ENDIF
ENDIF
RELEASE WINDOW prin
POP KEY
RETURN

PROCEDURE f7 &&-----Процедура вызова меню подразделений
PUSH KEY CLEAR && Отключение клавиш
ACTIVATE POPUP podr && Активация меню подразделений PODR
IF LASTKEY()#27 && Если не нажата клавиша Escape,
  REPLACE a.podr WITH b.podr && в базе запоминается выбор
ENDIF
POP KEY && Восстановление клавиш
RETURN

PROCEDURE poisk &&-----Процедура поиска
PUSH KEY CLEAR
y=BAR() && Запоминается вид поиска
n=RECNO() && и номер текущей записи
DO CASE
CASE y=5 && Если "Отмена сортировки",
  SET ORDER TO 0 && отказ о главного индекса
CASE y=4 && Если поиск "По подразделению":
  SET ORDER TO 4 && активация индекса KADRPODR.IDX,
  ACTIVATE POPUP podr && вызов меню подразделений,
  a=PROMPT() && в переменную A заносится выбор,
  d=a && запрос запоминается
OTHERWISE && Остальные выборы
  SET ORDER TO y && Активация нужного индекса
  ACTIVATE WINDOW poisk && Активация окна задания ключа
DO CASE
CASE y=1 && Поиск "По фамилии"
  @ 0,0 GET a DEFAULT SPACE(25) && Задание фамилии
  READ
  a=zag(ALLTRIM(a),.f.) && удаление пробелов,конвертирование
  d=a
CASE y=2 && Поиск "По табелю"
  @ 0,0 GET a PICTURE '999' DEFAULT 0 && Задание табеля
  READ
  d=STR(a,3) && Сохранить запрос
CASE y=3 && Поиск "По средней зарплате"
  @ 0,0 GET a PICTURE '9999.99' DEFAULT 0 && Задание зарплат
  READ
ENDCASE
ENDCASE
IF y#5.AND.!EMPTY(a).AND.!SEEK(a).AND.y#3 &&Если выбран Поиск и
&& переменная A не пуста и поиск неудачный и не по зарплате,
WAIT 'Поиск '+PROMPT()+': '+D+' НЕ УДАЧНЫЙ' WINDOW
GO n && выдается сообщение и возврат на старую запись
ENDIF
DEACTIVATE WINDOW poisk
POP KEY
DEACTIVATE POPUP
RETURN

```



```

FUNCTION zag    &&-----Функция конвертирования букв в заглавные
PARAMETER p,k
PUSH KEY
s='абвгдежзийклмнопрстуфхцчшщъыьэюя' && Переменные для преобразо-
z='АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ' && ваний в заглавные буквы
IF k && Если это поле базы,
REPLACE &p WITH CHRTRAN(&p,s,z) && делается замещение поля
ELSE && Если переменная,
POP KEY
RETURN CHRTRAN(p,s,z) && выполняется присвоение
ENDIF
POP KEY
RETURN &&-----Конец модуля KADR1.PRG-----

```

Программа KADR2.PRG В этой программе используется окно редактирования, созданное с помощью команд @ ... SAY ... GET/EDIT и READ, а также реализуется поиск по сложному ключу. Файл базы данных KADR.DBF не индексируется.

Рассмотрим программу, обращая внимание только на новые элементы. После открытия баз данных и определения меню описываются окна вывода: основное окно KADR — для вывода всех полей базы данных, окно POISK — для задания ключа поиска. Окно STAT, занимающее одну 23-ю строку экрана, предназначено для вывода нужных пользователю служебных сообщений и имитирует статус-строку. Вообще-то никакой необходимости организовывать статус-строку именно в форме окна в данном примере нет — ее можно вывести и на экран. Однако при выдаче более значительных по объему сообщений окно предпочтительнее. Кроме того, если предполагается вывести на экран еще какую-то информацию, она затрет такую статус-строку.

Объявляется (PUBLIC) переменная U, которая формируется в процессе задания запроса на поиск. Это необходимо для того, чтобы она была доступна в двух разных процедурах — поиска (F3) и продолжения поиска (F5).

Если база пуста, она дополняется одной записью.

Далее производятся клавишные назначения. Клавишам Ctrl-U можно было бы приписать не процедуру, а саму команду упаковки PACK. Однако, поскольку упаковка обычно длительный процесс, здесь все-таки вызывается процедура UPAK, в которой пользователю, чтобы он не скучал и ему не казалось, что компьютер стоит, выводится сообщение "Ждите — идет упаковка данных" и текущее время (проектируется на статус-строку). Так как вывод времени не зависит от состояния компьютера и ведется непрерывно, у пользователя не возникает впечатления, что он не работает. Ранее в начале программы по команде SET HOUR TO 24 устанавливается привычный нам 24-часовой формат времени.

Непосредственно ввод данных выполняется в процедуре EKRAN, которая формирует READ-окно, изображенное на рис.22.2.

В заголовке окна выводится текущая дата (DATE()). Ввод фамилии как в основном окне, так и в окне задания ключа поиска осуществляется с помощью форматной функции конвертирования символов строки в заглавные буквы FUNCTION '!' (считаем, что система реализует эту функцию для русских букв). Ввод даты рождения ограничен только трудоспособным возрастом 16 — 80 лет. Ввод пола и семейного положения теперь возможен с использованием функции ввода M. Это очень удобно, поскольку выбор нужного значения осуществляется просто нажатием на клавишу Space (Пробел), о чем есть подсказка в опции MESSAGE. Содержимое мемо-поля PER (перемещения) предъявляется командой @ ... EDIT, которая не требует для доступа к мемо-полю нажатия клавиш Ctrl-Home. С тем чтобы пользователю было легче ориентироваться в этом поле, выше команды EDIT сформированы заголовки колонок. В нижней части окна указаны управляющие клавиши, кроме клавиш

PgUp/PgDn, действие которых очевидно.

После нажатия любой клавиши из клавишного меню, вызывающей движение в базе, вызывается процедура DVIG. Ее код запоминается в переменной R (R=LASTKEY()).

ДАННЫЕ О СОТРУДНИКЕ Сегодня 02.10.92			
фамилия, инициалы СИДОРОВ П.С.			
дата рождения (день.месяц.год) 12.10.56			
табельный номер 13	количество детей 1	пол (М или Ж) М	
семейное положение X		ср. зарплата 350.00	
подразделение ОГМ			
Перемещения по службе:			
Дата зачисл.	Дата увольн.	Подразделение	Должность
01.01.85	20.06.78	Литейный цех	рабочий
21.06.78	05.12.90	Отдел гл.механика	рабочий
06.10.91		Отдел гл.механика	мастер
~PgDn-конец ~PgUp-начало ~T-удал. ~N-доп. ~U-сжатие F3-поиск Выход с сохранением изменений - Ctrl-End, Без - Esc			

Рис.22.2

Если R=3, значит, была нажата клавиша PgDn и нужно перейти к следующей записи. Однако вблизи конца и начала базы следует соблюдать осторожность. Перемещаться вниз (SKIP) можно, только если еще не достигнут конец файла (!EOF()). Если он достигнут, нужно вернуться (GO BOTTOM) на последнюю запись. Одновременно выдается сообщение о конце данных.

Если R=18 — нажата клавиша PgDn. Действия здесь аналогичны.

Если R=31 (Ctrl-PgUp) — переход к началу базы (GO TOP).

Если R=30 (Ctrl-PgDn) — переход к концу базы (GO BOTTOM).

Для обновления содержимого экрана используется команда SHOW GETS.

По достижении конца или начала базы, выполнении пометки записи, предназначенной для удаления, и т.д. в статус-строке выводятся соответствующие сообщения. Для этого вызывается пользовательская функция STAT(), куда, в качестве аргумента передается текст сообщения или пустой символ (""), если строку нужно очистить. В самой функции STAT активируется окно STAT, в центре которого выводится текст и снова активируется главное окно KADR, поскольку статус-строка "обслуживает" только это окно. Если доступ к ней нужно организовать из нескольких окон, при входе в процедуру нужно запомнить имя текущего окна (WONTOP()), с тем чтобы затем активировать именно его.

Процедура F7 по-прежнему активирует меню подразделений, однако здесь предусмотрено, чтобы меню вызывалось только в случае, если курсор находится в поле, для которого оно предназначается. Для этого в процедуре анализируется имя поля (VARREAD()), из которого произошел вызов. Если это PODR — меню активируется. Вызов меню может быть сделан и для поисковой переменной (см. ниже) PODR1. Поскольку ее имя частично совпадает с ключом PODR, то равенство VARREAD()='PODR' будет справедливо и здесь, ведь сравнение данных в FoxPro ведется по длине второго операнда. Далее в зависимости от того, из какого объекта произошел вызов, делается замещение или присваивание.

Процедура CTRLT обрабатывает нажатие клавиши Ctrl-T и помечает текущую запись на удаление с выдачей сообщения "Удалено". Снятие пометки производится той же клавишей.

Процедура CTRLN реагирует на нажатие Ctrl-N и добавляет новую пустую запись в базу.

Процедура F3 запускает процесс задания ключа поиска и самого поиска.

Здесь подавляется действие всех других клавиш, кроме F7 вызова меню подразделений, которое может понадобиться при поиске по полю Подразделение (PODR).

В процедуре реализуется поиск практически по всем полям базы, кроме SZAR, причем в случаях, где это оправдано, может быть задан диапазон поиска.

Для формирования критерия поиска создается ряд поисковых переменных, которые первоначально очищаются, и при этом им придаются тип и длина, соответствующие полям файла данных. Для удобства этим переменным даны имена, аналогичные именам полей, но заканчивающиеся цифрами 1 и 2: FAM1 — для задания разыскиваемой фамилии длиной 25 пробелов (SPACE(25)), TAB1, POL1, SEM1 — для указания табельного номера, пола и семейного положения, DTR1 и DTR2 — для нижней и верхней границ даты рождения и т.д. Две последние переменные должны иметь тип дата. Кроме того, вводится переменная, где будет формироваться выбор пользователя U.

Далее запоминается номер текущей записи N и активируется окно задания критерия поиска, где пользователю предлагается заполнить все или часть переменных в любой комбинации (рис.22.3).

ДАННЫЕ О СОТРУДНИКЕ Сегодня 28.11.92

фамилия, инициалы СИДОРОВ П.С.

дата рождения (день.месяц.год) 12.10.56

табельный номер 13 количество детей 1 пол (М или Ж) М

ДАННЫЕ ДЛЯ ПОИСКА

фамилия [] табель 0

родился от [] до [] детей от 0 до 0

Подразделение - F7 []

пол (М/Ж) [] сем. положение (Б/Х/Р) []

Начать поиск - PgUp, Отказ - Esc

Выход с сохранением изменений - Ctrl-End, Без-Esc

Дата зачи 01.0 21.0

~PgD

Рис.22.3

Здесь несколько изменен ввод в поле POL1 по сравнению с POL. Среди разрешенных значений М и Ж указан и пробел (просто запятая перед буквой М). Это нужно, если поиск по данной переменной не предполагается и она должна остаться пустой. Аналогичным образом построен ввод в поле SEM1. Для переменной PODR1 нажатием клавиши F7 может быть вызвано меню подразделений.

Если не нажата клавиша Escape (LASTKEY()#27), т.е. не произошел отказ от поиска, анализируются вводы, сделанные пользователем в окне поиска.

Переменные, которые он при вводе пропускает, в критерий (переменную U) не включаются. Как это происходит? В программе создается символьное изображение критерия поиска в переменной U, которая первоначально получает символьное значение '.Т.' (напоминаем, что .Т. без апострофов означает логическое значение "Истина"). В зависимости от ввода задаваемых поисковых переменных к переменной U добавляются фразы вида

".AND. <УСЛОВИЕ ПОИСКА>".

Сначала анализируется поисковая переменная FAM1 (фамилия):

```
u=u+IIF(EMPTY(fam1),'','.AND.fam= ['+ALLTRIM(fam1)+''])
```

Если она не была введена, т.е. осталась пустой (EMPTY(fam1)), то U не изменится (U='.Т.+' — к переменной U добавляется пустая строка длиной ноль символов). В противном случае в критерий U включается указание на

поиск по фамилии, что даст следующее выражение:

```
u='.T..AND.FAM= [' + ALLTRIM(FAM1) + ' ]'
```

В полученной переменной строкового типа выражение ALLTRIM(FAM1) будет замещено на фактическое значение заданной фамилии, например если эта фамилия "ПЕТРОВ", то окончательно

```
u='.T..AND.FAM=[ПЕТРОВ]'
```

Квадратные скобки здесь употреблены вместо апострофов, поскольку последние уже участвуют в образовании строки.

Освобождение поисковой переменной от концевых пробелов, если они есть, как уже говорилось, позволяет вести поиск и по неполной фамилии. Поскольку в FoxPro сравнение символьных переменных идет по длине второго операнда, первым в отношении должно быть указано имя поля (более длинная строка), а затем — имя переменной:

```
<ИМЯ ПОЛЯ> = ALLTRIM(<ИМЯ ПЕРЕМЕННОЙ>)
```

При анализе дат сначала также проверяется факт ввода данных. Если DTR1 (нижняя дата рождения) не пусто, к критерию поиска присоединяется строка

```
'.AND.DTR>={' + DTOC(DTR1)+ '}'
```

Поскольку для присоединения данные должны быть символьного типа, то переменная DTR1 сначала преобразуется в строку (DTOC()) и включается в выражение (+). Затем она обрамляется фигурными скобками для того, чтобы в итоге все-таки получился тип дата. Так, если задана дата (DTR1) рождения 05.08.70, то ключ поиска дополнится следующим выражением:

```
'.AND.DTR>={05.08.70}'
```

Аналогичным образом трансформируется верхняя дата рождения DTR2.

Если нужно найти работника, который родился в 1960 г., то в ответ на запрос о датах следует указать

```
Родился от 01.01.60 до 31.12.60
```

Если ввести только первую дату, будут найдены все записи для лиц, родившихся в 1960 г. и позже. Если только вторую — то все, родившиеся в 1960 г. и раньше. Если нужен поиск по фиксированной дате, значения DTR1 и DTR2 должны совпадать.

Поиск по табельному номеру, полу и семейному положению может быть только точным, и поэтому в соответствующих условиях стоит знак "=".

При включении в переменную U табельного номера он превращается сначала в строковый тип (STR()).

Например, если FAM1='ПЕТРОВ', DTR1={05.08.70}, TAB1=38 и POL='М', переменная U получит следующее значение:

```
u='.T..AND.FAM=[ПЕТРОВ].AND.DTR>={05.08.70}.AND.TAB= 38.AND.POL=[М]'
```

Замечание. Можно было бы построить выражение U более простым способом, оставив в критерии имена переменных, а не их фактические значения. Например, для рассмотренного случая достаточно получить

```
u='.T..AND.FAM=FAM1.AND.DTR>=DTR1.AND.TAB=TAB1.AND.POL=POL1''
```

Более сложный способ выбран здесь потому, что критерий поиска нужен не только в данной процедуре F3, но и в процедуре продолжения поиска F5. В этом случае необходимо, чтобы в F5 были доступны не только переменная U, но и все переменные, участвующие в образовании критерия (FAM1, DTR1 и

т.д.). Хотя это можно сделать, объявив их все глобальными (PUBLIC), мы избрали такой способ, поскольку это избавляет нас от необходимости помнить о них в дальнейшем.

После того как строка U сформирована, к ней применяется функция макроподстановки &, которая в данном случае освобождает ее от апострофов, т.е. преобразует тип переменной U из символьного в логический. Таким образом, команда LOCATE FOR &U эквивалентна LOCATE FOR .T..AND. ... Символ .T. теперь приобретает смысл логического значения "Истина". Он никак не влияет на анализ следующих условий и введен только для того, чтобы строка критерия была корректной. Поскольку все группы присоединяемых в дальнейшем условий начинаются с .AND. (логическое И), самому первому .AND. должно предшествовать какое-то другое логическое выражение. Удобно взять его равным .T., что сделает всю строку синтаксически верной, но не изменит ее сути. Можно поступить и наоборот. Взять исходное значение переменной U пустым (''), а все элементы, включаемые в условие, замкнуть словом .AND.. В этом случае по завершении формирования переменной U необходимо удалить из него одно последнее лишнее .AND..

Далее проверяется содержимое переменной U. Если оно не было заполнено (в точности равно .T.), поиск не ведется. В противном случае находится первая удовлетворяющая ему запись (LOCATE FOR &U).

Рассмотренная универсальная техника формирования сложного критерия поиска стала возможной благодаря наличию в языке FoxPro такого полезного средства, как макроподстановка. Здесь допускается использовать функцию подстановки EVALUATE(), т.е. команды LOCATE FOR EVALUATE(U). Ту же самую задачу можно решить, применив команды IF и/или DO CASE, но это будет исключительно сложно.

Если поиск оказался неудачным, происходит возврат на исходную запись (GO N) с выдачей соответствующего сообщения. Если нужная запись найдена, в статус-строке выводится фраза "Продолжение поиска — F5", говорящая о возможности продолжения поиска при нажатии клавиши F5, и обновляется главное окно — оно будет отображать запись, найденную по команде LOCATE.

Процедура F5 осуществляет (CONTINUE) продолжение поиска, если ранее он был удачным и переменная U не пуста (не равна .T.).

```
*-----Программа KADR2.PRG (нужны файлы: KADR.DBF и PODR.DBF)-----
SET DATE GERMAN          && Установка операционной среды системы
SET TALK OFF
SET DELETE ON
SET ESCAPE OFF
SET HELP OFF
CLEAR MACROS
SET HOUR TO 24           && Устанавливается 24-часовой формат времени
CLEAR
USE kadr IN a
USE podr IN b
DEFINE POPUP podr FROM 0,4 PROMPT FIELD b.podr
ON SELECTION POPUP podr DEACTIVATE POPUP
DEFINE WINDOW kadr FROM 5,7 TO 18,70;
    TITLE 'ДАННЫЕ О СОТРУДНИКЕ Сегодня '+DTOC(DATE());
    FOOTER 'Выход с сохранением изменений - Ctrl-End, Без-Esc'
DEFINE WINDOW poisk FROM 9,12 TO 18,72 SHADOW;
    TITLE 'ДАННЫЕ ДЛЯ ПОИСКА' COLOR SCHEME 7;
    FOOTER 'Начать поиск - PgUp, Отказ - Esc' && Окно поиска
DEFINE WINDOW stat FROM 23,0 TO 23,79 NONE COLOR n*/w
PUBLIC u                  && Переменная условий поиска
IF EOF()                  && Если база пуста - дополнение
    APPEND BLANK
ENDIF
ON KEY LABEL ctrl+t DO ctrlt
ON KEY LABEL ctrl+n DO ctrln
```



```

ON KEY LABEL ctrl+u DO upak
ON KEY LABEL f3 DO f3
ON KEY LABEL f5 DO f5
ON KEY LABEL f7 DO f7
ON KEY LABEL pgup DO dvig
ON KEY LABEL ctrl+pgup DO dvig
ON KEY LABEL pgdn DO dvig
ON KEY LABEL ctrl+pgdn DO dvig
DO ekran
CLEAR WINDOWS
ON KEY                                &&-----Конец головной программы

PROCEDURE ekran                      &&-----Процедура отображения экрана
ACTIVATE WINDOW stat NOSHOW
ACTIVATE WINDOW kadr NOSHOW
@ 0, 8 SAY 'фамилия, инициалы ' GET fam FUNCTION '!'
@ 1,10 SAY 'дата рождения (день.месяц.год)';
      GET dtr RANGE DATE()-80*365, DATE()-16*365
@ 2, 1 SAY 'табельный номер' GET TAB
@ 2,23 SAY 'количество детей' GET det
@ 2,44 SAY 'пол (М или Ж)' GET pol FUNCTION 'М М,Ж';
      MESSAGE 'Выбор клавишей ПРОБЕЛ'
@ 3, 5 SAY 'семейное положение' GET sem FUNCTION 'М Б,Х,Р';
      MESSAGE 'в браке, Холост, Разведен - выбор клавишей ПРОБЕЛ'
@ 3,32 SAY 'ср. зарплата' GET szar
@ 4,16 SAY 'подразделение' GET podr;
      MESSAGE 'выбор подразделения - F7'
@ 5, 0 SAY PADC('Перемещения по службе:',62,'-') COLOR N/G
@ 6, 0 SAY 'Дата' |Дата| Подразделение |'+;
      'Должность' |COLOR N/G
@ 7, 0 SAY 'зачисл. |увольн. |'+;
      |COLOR N/G
@ 8, 0 SAY REPLICATE('-',62) COLOR N/G
@ 9, 0 EDIT per COLOR SCHEME 4 SIZE 3,62;
      MESSAGE 'Выход по Tab и Ctrl-Tab'
@ 12, 0 SAY '~PgDn-конец ~PgUp-начало'+;
      '~T-удал. ~N-доп. ~U-сжатие F3-поиск'
SHOW WINDOW kadr,stat
READ CYCLE
RETURN

FUNCTION stat                        &&-----Функция индикации статус-строки
PARAMETERS a
ACTIVATE WINDOW stat                && Активация окна статус-строки
?PADC(a,79)                         && Центрирование и выдача сообщения
ACTIVATE WINDOW kadr                && Активация основного окна ввода
RETURN

PROCEDURE dvig                      &&-----Процедура перемещения в базе
PUSH KEY CLEAR
r=LASTKEY()                         && Запоминание нажатия
=stat('')                           && Очистка статус-строки
DO CASE
  CASE r=3.AND.!EOF()                && Клавиша PgDn -
    SKIP                             && движение вниз
    IF EOF()                         && Если теперь конец -
      GO BOTTOM                       && назад к последней записи
      =stat('Конец данных')         && Вывод в статус-строке
    ENDIF
  CASE r=18.AND.!BOF()                && Клавиша PgUp
    SKIP -1
    IF BOF()
      =stat('Начало данных')
    ENDIF
    GO TOP
  CASE r=31                           && Клавиши ~PgUp
    GO TOP
    =stat('Начало данных')
  CASE r=30                           && Клавиши ~PgDn
    GO BOTTOM
    =stat('Конец данных')
ENDCASE
SHOW GETS
POP KEY
RETURN

```



```

PROCEDURE f7                &&-----Процедура вызова меню подразделений
PUSH KEY CLEAR
IF VARREAD()='PODR'        && Если имя перем./поля начинается на PODR,
    ACTIVATE POPUP podr    && активируется меню подразделений
    IF VARREAD()='PODR'    && Если это поле БД PODR,
        REPLACE a.podr WITH b.podr    && в базе запоминается выбор
    ELSE                    && Если это переменная PODR1,
        podr1=b.podr        && ей делается присваивание
    ENDIF
ENDIF
POP KEY
RETURN

```

```

PROCEDURE ctrln             &&-----Процедура клавиши ^N - дополнение
PUSH KEY CLEAR
APPEND BLANK
=stat(' Новая запись')
SHOW GETS
POP KEY
RETURN

```

```

PROCEDURE ctrlt             &&-----Процедура клавиши ^T - удаление
PUSH KEY CLEAR
IF DELETE()                && Если запись помечена,
    RECALL                  && пометка снимается
    =stat('')
ELSE                        && Если нет, помечается
    DELETE
    =stat('Удалено')
ENDIF
POP KEY
RETURN

```

```

PROCEDURE f3                &&-----Процедура поиска при нажатии клавиши F3
=stat('')
PUSH KEY CLEAR              && Отмена всех клавиш
ON KEY LABEL f7 DO f7       && Восстанавливается действие F7
n=RECNO()                   && Запоминается номер записи
ACTIVATE WINDOW poisk       && Активируется окно поиска
&& Отображаются данные, вводимые для поиска
@ 1,3 SAY 'фамилия' GET fam1 FUNCTION '!' DEFAULT SPACE(30)
@ 1,COL()+4 SAY 'табель' GET tab1 PICTURE '999' DEFAULT 0
@ 3,4 SAY 'родился от' GET dtr1 DEFAULT {}
@ 3,COL()+1 SAY 'до' GET dtr2 DEFAULT {}
@ 3,COL()+5 SAY 'детей от' GET det1 PICTURE '9' DEFAULT 0
@ 3,COL()+1 SAY 'до' GET det2 PICTURE '9' DEFAULT 0
@ 5,12 SAY 'Подразделение - F7' GET podr1 DEFAULT SPACE(15)
@ 7,6 SAY 'пол (М/Ж)' GET pol1 FUNCTION 'М,М,Ж' DEFAULT ' '
@ 7,COL()+9 SAY 'сем. положение (Б/Х/Р)';
GET sem1 FUNCTION 'М,Б,Х,Р' DEFAULT ' '

```

```

READ
ON KEY
DEACTIVATE WINDOW poisk
IF LASTKEY()#27             && Если не нажата клавиша Escape
    u='.t.'
    && Если введена фамилия, она включается в переменную U
    u=u+IIF(EMPTY(fam1), ' ', AND.fam = ['+ALLTRIM(fam1)+'])
    u=u+IIF(EMPTY(dtr1), ' ', AND.dtr=>{' +DLOC(dtr1)+'}) && родился от
    u=u+IIF(EMPTY(dtr2), ' ', AND.dtr<=>{' +DLOC(dtr2)+'}) && родился до
    u=u+IIF(EMPTY(det1), ' ', AND.det>=' +STR(det1,2)) && детей от
    u=u+IIF(EMPTY(det2), ' ', AND.det<=' +STR(det2,2)) && детей до
    u=u+IIF(EMPTY(tab1), ' ', AND.tab = ' +STR(tab1,3)) && табель
    u=u+IIF(EMPTY(pol1), ' ', AND.pol = [' +pol1+']) && пол
    u=u+IIF(EMPTY(sem1), ' ', AND.sem = [' +sem1+']) && сем. полож.
    u=u+IIF(EMPTY(podr1), ' ', AND.podr= ['+ALLTRIM(podr1)+']) && подразд.
    *-----Выражение поиска сформировано в переменной U-----
    IF u='.t.' && Если ничего не введено - конец
    ELSE && Иначе -
        LOCATE FOR &u && ведется поиск
        IF !FOUND() && Если он не удачный,
            GO n && возврат на старую запись,
            u='.t.' && сброс поисковой переменной,
            WAIT 'Не найдено' WINDOW && выдается сообщение
        ELSE && Если найдено, выводится сообщение

```



```

      =stat('Продолжение поиска - F5')
    ENDIF
  ENDIF
ENDIF
SHOW GETS
POP KEY
RETURN

PROCEDURE upak      &&-----Процедура упаковки базы
PUSH KEY CLEAR
WAIT 'Ждите идет упаковка данных' WINDOW NOWAIT && Сообщение
=stat('')
SET CLOCK TO 23,35      && Отображаются часы
PACK                   && Упаковка базы
WAIT CLEAR             && Удаление сообщения
SET CLOCK OFF          && Удаление часов
SHOW GETS
POP KEY
RETURN

PROCEDURE f5      &&----- Процедура продолжения поиска по F5
PUSH KEY CLEAR
n=RECNO()           && Запоминается номер текущей записи
IF 'u==' . t.      && Если переменная условий непуста
  CONTINUE         && поиск продолжается
  IF !FOUND()      && Если поиск неудачный -
    =stat('')
    GO n           && возврат на старую запись
    u=' . t.'
    WAIT 'Не найдено' WINDOW      && Сообщение "Не найдено"
  ENDIF
ENDIF
SHOW GETS
POP KEY
RETURN &&-----Конец модуля KADR2.PRG-----

```

Программирование форматной функции М. Хотя в команде BROWSE (программа KADR1.PRG) и не предусмотрено непосредственное использование, как в команде

```
@ 9,54 SAY 'пол (М или Ж)' GET pol FUNCTION 'М М,Ж'
```

последнего примера, функции ввода М, когда выбор нового значения поля (из нескольких возможных) осуществляется нажатием клавиши Space, при желании этот очень удобный механизм можно запрограммировать. Одно из возможных решений приведено ниже для поля, содержащего сведения о семейном положении (SEM), которое может иметь только значения "Б", "Х", "Р" и " ", если семейное положение неизвестно. Как и ранее, выбор осуществляется нажатием клавиши Space.

Поскольку клавиша Пробел интенсивно используется при редактировании, она только временно закрепляется за полем SEM в качестве средства выбора через опцию :W с помощью функции SEM(). Здесь же выводится сообщение об этом, а также указывается, что выбор в поле будет зафиксирован только после нажатия клавиши Enter. В переменной X запоминается текущее значение поля на случай, если придется отказаться от выбора.

За клавишей Пробел закрепляется процедура SEM, в которой производится циклическая подстановка новых значений при каждом нажатии. Чтобы новое значение поля было немедленно отображено на экране, в заключение процедуры BROWSE-окно обновляется. В качестве имени взято первое латинское слово ("Ctrl") в его заголовке (TITLE).

Выход из поля SEM осуществляется через функцию OTM(), где производится отмена закрепления процедуры за клавишей Space. Если выход из поля был инициирован нажатием клавиши Enter, программа считает, что пользователь отказался от изменения данных в поле, и присваивает ему прежние значение X.

```

SET TALK OFF
CLEAR
PUBLIC x
USE kadr
DEFINE WINDOW kadr FROM 1,1 TO 10,35 COLOR SCHEME 10
BROWSE FIELDS fam,sem :w=sem():v=otm():F;
        WINDOW kadr TITLE 'Ctrl-W - выход'
ON KEY

FUNCTION sem &&-----Функция входа в поле SEM
@ 15,10 SAY 'Space - выбор (Б/Х/Р), Enter - сохранение'
x=sem
&& Запоминание прежнего значения поля
ON KEY LABEL spacebar DO vib && Переназначение клавиши Space
RETURN

FUNCTION otm &&-----Функция выхода из поля SEM
ON KEY LABEL spacebar && Отмена назначения на клавишу Space
&& Если выход из поля произошел не с помощью клавиши Enter,
IF LASTKEY()#13
    REPLACE sem WITH x && возвращается прежнее значение поля
ENDIF
@ 15,10
RETURN

PROCEDURE vib &&-----Процедура выбора значения поля SEM
DO CASE
CASE sem='Б' && Ввод "по кругу" одного из
    p='X' && допустимых значений в
    && последовательности: 'X','P',' ','Б'
CASE sem='X'
    p='P'
CASE sem='P'
    p=' '
CASE sem=' '
    p='Б'
OTHERWISE
    p='Б'
ENDCASE
REPLACE sem WITH p && Занесение выбранного значения в поле
SHOW WINDOW CTRL REFRESH && Обновление окна
RETURN

```

Следующее, более универсальное решение приведено ниже. Здесь альтернативный выбор при нажатии клавиши Space предусмотрен в полях POL и PODR базы KADR.DBF. Для этого при входе, например в поле POL, вызывается одноименная функция POL(), в которой делается назначение на клавишу Space процедуры VIB, куда в качестве параметров передаются все возможные значения поля ("М" и "Ж"). Аналогично устроен ввод в поле PODR. Таким образом, при попадании в перечисленные поля при нажатии на клавишу Space можно обратиться к процедуре выбора VIB, принимающей до шести параметров — от X1 до X6. Большее количество альтернатив не имеет смысла перебирать таким образом — это быстрее сделать через POPUP-меню.

В переменной N фиксируется фактически переданное число параметров. Далее в цикле все они просматриваются и сравниваются с текущим значением поля. Если совпадение найдено, то в поле вводится (REPLACE) следующий по порядку параметр. Исключение составляет случай, когда значение поля V совпадает с самым последним переданным параметром. Тогда в поле пересылается первый (X1) параметр из списка. Если совпадение не обнаружено, в поле также пересылается первый элемент. Такая технология дает возможность пользователю циклически перебирать все возможные альтернативы ввода. По завершении выбора данных и выходе из текущего поля клавишное назначение отменяется в процедуре OTM() и клавиша Space освобождается.

Обратите внимание, что в программе интенсивно используются средства подстановки — макроподстановка &, функция EVALUATE() и круглые скобки (например, (VARREAD())).


```

SET TALK OFF
CLEAR
PUBLIC x
USE kadr
DEFINE WINDOW kadr FROM 1,1 TO 18,50 COLOR SCHEME 10
BROWSE FIELD fam,pol :W=pol() :V=otm() :F, podr :W=podr();
      :V=otm() :F TITLE 'Ctrl-W  выход' WINDOW kadr
ON KEY

FUNCTION pol      &&-----Функция входа в поле POL (пол)
x=pol
ON KEY LABEL spacebar DO vib with 'М','Ж'
@ 20,15 SAY 'Выбор клавишей Пробел (М/Ж)'      && Подсказка
RETURN

FUNCTION podr     &&-----Функция входа в поле поле PODR
x=podr
ON KEY LABEL spacebar DO vib with 'ОГМ','КБ','Бухгалтерия'
@ 20,15 SAY 'Выбор клавишей Пробел (ОГМ/КБ/Бухгалтерия)'
RETURN

FUNCTION otm      &&-----Функция выхода из полей POL и PODR
ON KEY LABEL spacebar
IF LASTKEY()#13      && Если выход из поля не по Enter -
  REPLACE (VARREAD()) WITH x && возвращается старое значение
ENDIF
@ 20,15      && Очистка экрана от подсказки
SHOW WINDOW CTRL REFRESH      && Обновление окна
RETURN

PROCEDURE vib      &&-----Процедура выбора значения поля
PARAMETERS x1,x2,x3,x4,x5      && Прием параметров
n=PARAMETERS()      && Число параметров
vv=ALLTRIM(EVALUATE(VARREAD()))      && Содержимое поля
FOR i=1 TO n      && Перебор параметров
  xx='x'+LTRIM(STR(i))      && Имя очередного параметра
  IF &xx=vv      && Если значения параметра и поля совпадают,
    && берется следующий по порядку или первый, если больше нет,
    xx='x'+LTRIM(STR(IIF(i=n,1,i+1)))
    REPLACE (VARREAD()) WITH &xx      && делается присвоение
    SHOW WINDOW CTRL REFRESH      && Обновление окна
  RETURN      && Выход
ENDIF
ENDFOR      && Если совпадений не найдено,
REPLACE (VARREAD()) WITH x1      && берется первый параметр
SHOW WINDOW CTRL REFRESH
RETURN

```

Совмещение BROWSE- и READ-окон. Очень удобным может оказаться совмещение BROWSE-окна и пользовательского окна, содержащего области ввода, сформированные командами @...GET и READ для одной и той же базы данных. Тогда в BROWSE-окно выносятся небольшое количество важнейших полей, которые позволяют ориентироваться в базе и как бы образуют ее оглавление, а в READ-окне отображаются все остальные поля, причем возможно их редактирование. Схема такой программы приведена ниже для базы KADR.DBF.

Ее интерфейс состоит из двух окон: окна KADRB для команды BROWSE (здесь отображаются только поля FAM и TAB) и окна KADRR для команд @...GET со всеми остальными полями. Оба окна интегрируются в общий интерфейс командой READ. Поскольку опция TITLE здесь не используется, именем BROWSE-окна будет имя базы данных "KADR". Для того чтобы данные в READ-окне постоянно обновлялись при перемещении курсора в BROWSE-окне, последнее через опцию WHEN вызывает функцию BR(), содержащую команду SHOW GETS WINDOW kadr.

```

SET TALK OFF
CLEAR

```

```

USE kadr
DEFINE WINDOW kadb FROM 1,0 TO 20,33 COLOR SCHEME 10
DEFINE WINDOW kadrr FROM 1,36 TO 20,78 COLOR SCHEME 1
r=RECNO()
BROWSE FIELDS k=IIF(r=RECNO(), '>', ''), ;
               fam :H='Фамилия', ;
               tab :H='Таб.' ;
               WINDOW kadb NOWAIT WHEN br() LOCK 1
ACTIVATE WINDOW kadrr
@ 0, 0 SAY 'Фамилия' GET fam
@ 2, 0 SAY 'Родился' GET dtr
@
READ CYCLE
RELEASE WINDOWS kadr, kadrr

FUNCTION br &&-----Функция обновления окон
r=RECNO()
SHOW GETS WINDOW kadb
SHOW WINDOW kadr REFRESH
RETURN

```

Кроме того, здесь реализована одна полезная функция. Поскольку при переходе в READ-окно курсор покидает BROWSE-окно, становится не видно, какой именно записи соответствуют данные в READ-окне. Ввиду этого в команду BROWSE введено вычисляемое поле K, которое дублирует курсор значком ">", если номер текущей записи равен некоторому числу R (первоначально R=RECNO()). При переходе в другую запись через опцию WHEN в функции BR() переменная R получает новое значение и окно KADR обновляется (SHOW WINDOW kadr) для удаления старого значка.

Программа KADR3.PRG. В заключение рассмотрим фрагмент программы, где сочетаются экран ввода данных и двухуровневое световое меню, которое работает как клавишное. Данный материал посвящен только этому моменту, поэтому обрабатывающая часть программы опущена.

Программа и меню (рис.22.4) приведены ниже. Сам экран ввода содержит только два поля — табельный номер и фамилию из базы KADR.DBF. Горизонтальное меню по управлению данными UPR имеет пять пунктов, из которых пункт "Сервис" содержит вспомогательное POPUP-меню SERV для выполнения редких или необратимых действий. Остальные пункты горизонтального меню содержат постоянно необходимые средства перемещения в базе.

Обращение к меню возможно двумя способами. Во-первых, нажатием клавиши F10 можно вызвать его целиком, а затем, перемещаясь в нем, выбрать нужный пункт и нажать клавишу Enter. Во-вторых, все элементы горизонтального меню могут быть вызваны отдельно нажатием "горячих" клавиш PgUp, PgDn, Ctrl-PgUp, Ctrl-PgDn, F3. При этом пункты, связанные с перемещением в базе, будут тут же выполнены без подтверждения нажатием клавиши Enter, а пункт "Сервис" сразу развернет вспомогательное меню. Такая технология позволяет организовать быстрый доступ к постоянно требующимся простым средствам и одновременно обратиться к более сложным меню.

Для того чтобы указать пользователю на возможности клавиши F10, в горизонтальном меню специально введен PAD-пункт UPR6 со строкой 'Вызов меню — F10'. Чтобы сделать этот пункт неактивным, он заблокирован опцией SKIP.

Однако меню в зависимости от способа первоначального предъявления может работать по-разному. Если использована команда ACTIVATE MENU upr NOWAIT, то меню ведет себя обычным образом. Оно постоянно доступно для мыши. В него можно попасть, нажав "горячие" клавиши. Например, чтобы перейти на следующую запись, следует сразу нажать клавишу PgDn. После выбора какого-то пункта меню курсор останется на месте.

Если использована команда SHOW MENU, то пункты меню являются, скорее, просто строками указаний на "горячие" клавиши, которые следует нажать, чтобы выполнить определенное действие. Курсор при этом останется в окне редактирования (кроме нажатия F3). Хотя и здесь в меню можно попасть нажатием клавиши F10, выбор любого его пункта немедленно возвращает нас назад.

Поскольку обе эти команды не могут присутствовать одновременно, команда SHOW MENU показана в программе как имеющая статус комментария (*). Завершение редактирования и выход из программы могут быть осуществлены нажатием клавиши Escape.

Вызов меню - F10	
Вверх PGUP	Вниз PGDN
Начало ^PGUP	Конец ^PGDN
Сервис F3	
Табельный номер 468	Упаковка
Фамилия МИРОНОВ Р.И.	Переиндексация
	Печать

Рис.22.4

```

*-----Программа KADR3.PRG-----
CLEAR MACRO
SET TALK OFF
SET ESCAPE OFF
CLEAR
USE kadr
DEACTIVATE MENU upr
RELEASE MENU upr
DEFINE WINDOW w FROM 3,3 TO 6,42
DEFINE MENU upr KEY F10 IN SCREEN NOMARGIN
DEFINE PAD upr1 OF upr PROMPT 'Вверх' KEY PgUp AT 1,1
DEFINE PAD upr2 OF upr PROMPT 'Вниз' KEY PgDn
DEFINE PAD upr3 OF upr PROMPT 'Начало' KEY Ctrl+PgUp, '^PgUp'
DEFINE PAD upr4 OF upr PROMPT 'Конец' KEY Ctrl+PgDn, '^PgDn'
DEFINE PAD upr5 OF upr PROMPT 'Сервис' KEY F3
DEFINE PAD upr6 OF upr PROMPT 'Вызов меню - F10';
SKIP COLOR SCHEME 12 AT 0,18
ON SELECTION MENU upr DO ppp
ON SELECTION PAD upr5 OF upr ACTIVATE POPUP serv

DEFINE POPUP serv IN SCREEN FROM 2,39
DEFINE BAR 1 OF serv PROMPT 'Упаковка'
DEFINE BAR 2 OF serv PROMPT 'Переиндексация'
DEFINE BAR 3 OF serv PROMPT 'Печать'
ON SELECTION POPUP serv DO ppp
ACTIVATE MENU upr NOWAIT      && Меню активно
*SHOW MENU upr                && Предъявление меню без активации
ACTIVATE WINDOW w             && Активация окна редактирования W
@ 0,3 SAY 'Табельный номер' GET tab
@ 1,3 SAY 'Фамилия' GET fam
READ CYCLE
DEACTIVATE MENU upr
RELEASE MENU upr
RELEASE WINDOW w
CLEAR
RETURN

PROCEDURE ppp      &&-----Процедура обработки выбора
DO CASE           && Разбор выбора из меню
CASE PROMPT()='Вверх'
SKIP -1
CASE PROMPT()='Вниз'

ENDCASE
SHOW GETS      && Обновление окна ввода
RETURN      &&-----Конец модуля KADR3.PRG-----

```


Завершая раздел, обобщим решения одного технического вопроса. Частично в примерах они уже использовались.

Находясь внутри поля базы данных, предъявляемого командой BROWSE/CHANGE или @...GET и READ, необходимо иметь возможность совершать управляющие действия, относящиеся только к данному полю. Для этого должны быть сделаны клавишные назначения, обрабатываемые, только когда курсор находится в нем. Такие действия могут быть реализованы с помощью опций W/WHEN и V/VALID (команд BROWSE и @ ... GET), контролирующих вход/выход в/из поле, и функции VARREAD(), возвращающей имя текущего поля (для команды BROWSE — первая буква имени прописная, остальные — строчные, для команды @...GET — все заглавные). Если процедура, в которой обрабатывается имя поля, задействована для обеих команд, можно перейти только к заглавным буквам, использовав функцию UPPER() (т.е. UPPER(VARREAD())).

Сделать клавишные закрепления за полем можно несколькими способами.

1. Закрепление процедуры за клавишей командой ON KEY LABEL и создание процедуры, в которой в зависимости от имени поля выполняются те или иные действия. Схема такой программы для команды BROWSE приведена ниже.

```
ON KEY LABEL <клавиша> DO <процедура>
BROWSE FIELD <поле1>, <поле2>
```

```
PROCEDURE <процедура>
DO CASE
CASE VARREAD()=<имя поля1>
    <обработка нажатия в поле1>
CASE VARREAD()=<имя пол2>
    <обработка нажатия в поле2>
ENDCASE
RETURN
```

Таким образом, можно с помощью одной <клавиши> и одной <процедуры> обрабатывать несколько полей. Нажатие <клавиши> в <поле>, для которого не предусмотрено никаких действий, останется без последствий, поскольку в <процедуре> этому полю не соответствует никакое CASE-условие. При желании, конечно, можно задействовать и несколько клавиш, и несколько процедур.

2. "Персональное" закрепление клавиш за полями. Здесь при входе в поле через опции W/WHEN в процедуре-функции, которую мы назовем <функциейW>, делается закрепление за некоторой клавишей <процедуры> обработки поля, а при выходе через опции V/VALID в другой <функцииV> — отмена закреплений. Схема программы приведена ниже.

```
BROWSE FIELD <поле1> :W=<функцияW> :V=<функцияV> :F
```

```
FUNCTION <функцияW>                                && Функция закрепления клавиши
ON KEY LABEL <клавиша1> DO <процедура>
RETURN
```

```
FUNCTION <функцияV>                                && Функция отмены закрепления
ON KEY LABEL <клавиша1>
RETURN
```

```
PROCEDURE <процедура>                                && Процедура обработки поля
<команды обработки поля>
RETURN
```

Чтобы отмена клавиши гарантировалась всегда, опция V команды BROWSE усилена ключом F.

3. Имитация попытки выхода из поля, с тем чтобы через опции V/VALID вызвать функцию обработки. Для этого необходимо, чтобы какая-то клавиша генерировала код выхода, например код клавиш Enter или Tab, но

распознавалась бы правильно функцией LASTKEY(). Сами клавиши выхода остаются в прежней роли, поскольку должна быть сохранена нормальная возможность выхода из поля. Имитировать одну клавишу другой можно с помощью команды SET FUNCTION TO, в которую включена команда KEYBOARD(), содержащая код клавиши выхода. Тогда в опции V/VALID с помощью функций IIF() и LASTKEY() можно выяснить, что же фактически было нажато, и, если это заданная клавиша, выполнить необходимую процедуру-функцию. Схема программы приведена ниже. Здесь клавиша F3 генерирует код клавиши Enter.

```
SET FUNCTION F3 TO KEYBOARD '{ENTER}'
BROWSE FIELD <поле1> :V=IIF(LASTKEY()=-2,<функция>,.T.)

FUNCTION <функция>                                && Функция обработки <поля1>
<обработка поля>
RETURN 0
```

Для того чтобы избежать выхода из <поля> при нажатии клавиши F3, <функция> завершается командой RETURN с аргументом 0. Нормальный выход из <поля> остается по-прежнему возможным, так как, если нажата, например, клавиша Enter/Tab, функция IIF() вырабатывает значение .T..

Здесь клавиша F3 может быть использована для нескольких <полей>, каждое из которых обрабатывается своей функцией.

Команда BROWSE имеет одну особенность. Если в <функции> изменяется содержимое <поля> командой REPLACE <поле> WITH <новое значение>, то новое содержимое поля появится на экране только после того, как мы переместим курсор в другое поле/запись. В этом случае перемещение можно сделать автоматическим, задав в команде RETURN аргумент .T.; либо вместо команды REPLACE воспользоваться командой KEYBOARD <новое значение>, которая "выдавит" из буфера клавиатуры в текущее поле новое содержимое, начиная с текущей позиции курсора; либо использовать команду SHOW WINDOW REFRESH, обновляющую содержимое BROWSE-окна независимо от положения курсора.

22.2. Древовидная организация данных

Взаимодействие данных, которое мы хотим отобразить с помощью компьютера, часто носит древовидный характер, когда один элемент данных связан с несколькими другими и число их обычно заранее неизвестно. Поскольку реляционные СУБД не позволяют сделать это естественным образом, приходится прибегать к программированию подобных связей.

Такое взаимодействие есть, например, в системе КАДРЫ. Здесь каждому человеку может соответствовать несколько строк о его перемещениях по службе. Связь базы данных с мемо-полем не вызывает проблем, но сами данные в мемо-поле никак не структурированы и, следовательно, очень неудобны для поиска, выборочного доступа и обработки. Для СУБД все мемо-поле представляет собой одну запись неопределенной длины. Абстрактную длину строки в мемо-поле определяет команда SET MEMOWIDTH. Чтобы получить доступ к какому-то фрагменту мемо-поля, нужно его найти с помощью функций поиска, которые, конечно, слишком медленные, и в большинстве случаев этих возможностей совершенно недостаточно.

В программе KADR2.PRG сделана попытка как-то структурировать мемо-поле PER, используя заголовки над окном @ ... EDIT. Таким же образом можно организовать и окно для мемо-поля, включив в опцию TITLE соответствующие заголовки.

Такое представление данных хотя и лучше, чем совершенно

неструктурированное, все же не допускает какой-либо входной контроль, вызов процедур, поиск и т.д. и, кроме того, дает пользователю большой простор для ошибок ввода. Например, смещение позиции ввода подстроки даты на одну колонку в последствии не позволит правильно извлечь ее из мемо-поля и превратить в тип дата.

Если требуется более жесткое структурирование данных, можно прибегнуть к перенесению всего мемо-поля во вспомогательную базу данных, из которой можно их взять в окно редактирования, например BROWSE-окно. Данные из мемо-поля теперь можно обрабатывать только здесь, после чего они возвращаются назад — в мемо-поле.

Ниже приведен фрагмент программы, реализующей задачу, в которой данные перемещаются следующим образом:

мемо-поле PER --> файл PER.DBF --> мемо-поле PER

Здесь база данных PER.DBF используется для взятия и обработки данных из мемо-поля. Ее структура (рис.22.5) соответствует желаемой структуре мемо-поля.

База PER.DBF

Содержание	Имя	Тип	Длина
Дата зачисления	DZ	Date	8
Дата увольнения	DU	Date	8
Должность	DOL	Character	20
Подразделение	POD	Character	15

Рис.22.5

Общая длина всех полей базы — 51 символ.

Данные из мемо-поля предъявляются командой BROWSE, например в форме рис.22.6.

Перемещения по службе			^End — сохранение
Зачислен	Уволен	Должность	Подразделение
02.07.82	04.10.89	Мастер	Цех N8
05.10.88	09.04.90	Старший инженер	Констр. бюро

Рис.22.6

Непосредственный доступ пользователя к мемо-полю теперь может быть совершенно исключен.

В программе MEMO1.PRG после основной части (первые восемь строк) имеется процедура MEMOPER для структурированной обработки мемо-поля PER, которая вызывается при нажатии клавиши F9. В процедуре описывается окно PER для предъявления данных, взятых из мемо-поля в базу PER.DBF. Важной командой здесь является SET MEMOWIDTH TO 51, по которой устанавливается длина строки 51 символ (по ширине базы PER.DBF).

В области С очищается (ZAP) от старых данных файл PER.DBF. Затем из мемо-поля PER (имеющего MEMLINES() строк) выделяется очередная строка P, из которой, в свою очередь, выделяются элементы нужной длины и заносятся в соответствующие поля базы PER.DBF. Поскольку все они могут быть только строкового типа, одновременно символы с первого по восьмой и с девятого по шестнадцатый конвертируются в тип дата. Далее открывается окно (BROWSE) редактирования базы PER.DBF.

После завершения редактирования при нажатии клавиши Ctrl-End выполняется обратная процедура — очищается мемо-поле и в него переносятся данные из базы PER.DBF.


```

*-----Программа MEMO1.PRG-----нужны файлы KADR.DBF, PER.DBF-----
SET SAFETY OFF
SET DELETE ON
USE kadr IN a
USE per IN c                                && Открытие базы PER.DBF
DEFINE WINDOW per FROM 6,22 TO 23,79        && Окно PER для мемо-поля
SET MEMOWIDTH TO 51                        && Установление ширины строки в мемо-поле
ON KEY LABEL f9 DO memoper                 && Вызов процедуры MEMOPER по F9
* Основное окно данных KADR.DBF
BROWSE TITLE 'F9 - перемещения'

PROCEDURE memoper                            &&-----Процедура обработки мемо-поля
PUSH KEY CLEAR
SELECT c
ZAP                                          && Очистка базы
FOR i=1 TO MEMLINES(a.per)                && Цикл по количеству строк
  p=MLINE(a.per,i)                        && Выделение очередной строки
  mdz=CTOD(SUBSTR(p,1,8))                  && Взятие элементов строки
  mdu=CTOD(SUBSTR(p,9,8))                  && в переменные
  mdol=SUBSTR(p,17,20)
  mpod=SUBSTR(p,37,15)
  APPEND BLANK                             && Заполнение базы PER.DBF
  REPLACE dz WITH mdz, du WITH mdu, dol WITH mdol, pod WITH mpod
ENDFOR
BROWSE WINDOW per;
  TITLE 'Перемещения по службе' 'End - сохранение';
  FIELD dz :H='Зачислен';
  du :H='Уволен';
  dol :H='Должность';
  pod :H='Подразделение';
  COLOR SCHEME 10                          && BROWSE-окно
IF LASTKEY()=23                            && Если выход по 'End',
  REPLACE a.per WITH ''                     && мемо-поле очищается и
  SCAN                                     && заполняется из базы PER.DBF
    REPLACE a.per WITH;
    DTOC(c.dz)+DTOC(c.du)+c.dol+c.pod ADDITIVE
  ENDSCAN
ENDIF
SELECT a
POP KEY
RETURN
*-----Конец модуля MEMO1.PRG-----

```

Приведенная программа позволяет обращаться с компонентами мемо-поля как с полями базы данных, включая любой контроль и преобразование. Также легко может быть осуществлен поиск в мемо-данных. Однако из-за необходимости перекачивать их через транзитную базу (ее лучше разместить на виртуальном диске), доступ к таким данным остается медленным, кроме того, для них не может быть выполнено индексирование.

Если рассмотренная технология не отвечает поставленным задачам, необходимо прибегнуть к хранению данных о перемещениях не в мемо-поле, а в постоянной вспомогательной базе данных. В этом случае мемо-поле PER вообще удаляется из базы KADR.DBF.

Назовем такую базу именем PEREM.DBF. Ее структура совпадает со структурой базы PER.DBF, если не считать дополнительное поле, посредством которого устанавливается связь записей вспомогательной базы PEREM.DBF и основной KADR.DBF. В этом поле должен храниться уникальный признак из записи KADR.DBF. Таким признаком может быть фамилия работника (поле FAM) и/или его табельный номер (поле TAB). Остановимся на табельном номере. Назовем это дополнительное поле в базе PEREM.DBF так же, как и в основной базе, TAB и определим для него идентичные тип и длину. Для установления связи с основной базой база PEREM.DBF проиндексирована по полю TAB — индексный файл PERTAB.IDX.

Создадим программу KADRPER.PRG, которая позволит на запрос пользователя о просмотре/заполнении базы перемещений PEREM.DBF, предъявлять BROWSE-окно, изображенное на рис.22.7.

ИВАНОВ А.А.		Должность	табель - 823	
Зачислен	Уволен		Подразделение	Табель
02.07.82	04.10.89	Мастер	Цех №8	823
05.10.88	09.04.90	Старший инженер	Констр. бюро	823

Рис.22.7

Эта программа приведена ниже. Здесь релизованы три процесса.

о Предъявление для любой записи из KADR.DBF всех записей из базы перемещений PEREM.DBF с тем же табельным номером для их редактирования/дополнения (клавиша F5).

о Вывод на экран содержимого базы KADR.DBF со всеми перемещениями из PEREM.DBF (клавиша F6).

о Упаковка обеих баз (клавиша F7).

Для отображения данных определяются два окна — KADR (для основной базы KADR.DBF) и PEREM (для вспомогательной базы PEREM.DBF и вывода данных на экран). Клавишное меню, вызываемое из команды BROWSE:

"F5 - перемещения F6 - вывод на экран F7 - упаковка"

реализовано командами вида ON KEY LABEL.

Процедура PEREM предъявляет вспомогательную базу PEREM.DBF по нажатию клавиши F5. Здесь сначала выясняется, есть ли в базе PEREM.DBF записи с табельным номером (b.tab), совпадающим с табельным номером из KADR.DBF (a.tab). Если нет — у пользователя запрашивается подтверждение запроса и, если не нажата клавиша Escape, база дополняется одной пустой записью, где табельный номер будет взят из базы KADR.DBF. В дальнейшем командой SET CARRY TO TAB устанавливается режим, когда любое дополнение базы повлечет копирование поля TAB в следующие записи. После этого в окне PEREM открывается BROWSE-окно редактирования вспомогательной базы. Чтобы пользователь не забыл, для кого он вызвал это окно, в его заголовке отображается фамилия и табельный номер работника (см. рисунок выше).

Процедура VIVOD осуществляет вывод данных из баз KADR.DBF и PEREM.DBF в окне PEREM. Для этого последовательно сканируется база KADR.DBF и выводится каждая ее запись. Затем в базе PEREM ищутся и выводятся все записи с совпадающими табельными номерами. Чтобы иметь возможность спокойного просмотра данных, в команду ? для каждой выводимой записи из базы KADR включена функция IIF, которая с помощью функции INKEY(0) создает паузу в случае, если достигнута нижняя граница окна (ROW()>=WROW()-1). Возобновление показа будет возможным теперь только при нажатии любой клавиши.

Процедура упаковки UPAK предусматривает удаление и непомеченных пользователем записей из вспомогательной базы PEREM.DBF, если соответствующая запись в основной базе помечена.

```

*-----Программа KADRPER.PRГ-----
*-----нужны файлы KADR.DBF, PEREM.DBF, PERTAB.DBF-----
CLEAR MACRO
SET TALK OFF
SET DELETE ON
SET ESCAPE OFF
CLEAR
USE kadr IN a
USE perem IN b INDEX pertab
DEFINE WINDOW kadr FROM 1,1 TO 18,79      && Основное окно KADR
DEFINE WINDOW perem FROM 3,7 TO 23,71     && Окно PEREM

```



```

ON KEY LABEL f6 DO vivid                                && Клавишные назначения
ON KEY LABEL f5 DO perem
ON KEY LABEL f7 DO upak
BROWSE WINDOW kadr COLOR SCHEME 10 TITLE;
'F5 - перемещения   F6 - вывод на экран   F7 - упаковка'

PROCEDURE perem      &&-----Процедура ввода Перемещений (F5)
PUSH KEY CLEAR
SELECT b
IF !SEEK(a.tab)      && Если в базе PEREM.DBF нет такого табеля
                     && - запрос дополнения базы
                     WAIT 'Дополнить Перемещения? (Esc - отказ)' WINDOW
                     IF LASTKEY()=27      && Если нажата клавиша Escape
                         POP KEY
                         RETURN            && - отказ от дополнения
                     ENDIF
                     APPEND BLANK        && В противном случае - дополнение базы
                     REPLACE b.tab WITH a.tab      && с копированием поля a.tab
ENDIF
SET CARRY TO tab      && Установление режима копирования поля TAB
BROWSE WINDOW perem KEY a.tab;
                     TITLE a.fam+' табель - '+STR(a.tab,3);
                     FIELD dz :H='Зачислен',;
                     du :H='Уволен',;
                     dol :H='Должность',;
                     pod :H='Подразделение',;
                     tab :H='Табель',;
                     COLOR SCHEME 10      && BROWSE-окно перемещений
SET CARRY OFF
SELECT a
POP KEY
RETURN

PROCEDURE vivid      &&-----Процедура вывода данных в окно PEREM
PUSH KEY CLEAR
SELECT a
ACTIVATE WINDOW perem
CLEAR
SCAN                && Сканирование основной базы KADR.DBF
                     && Вывод данных из KADR.DBF
                     ? fam,tab,IIF(ROW())>=WROW()-1.AND.INKEY(0)#0,',','')
                     SELECT b
                     IF SEEK(a.tab)      && Если в базе PEREM.DBF есть записи с тем-же
                                             && табелем, они выводятся
                     ? 'Зачислен Уволен Должность Подразд. Табель'
                     SCAN WHILE a.tab=b.tab
                     ? dz,du,dol,pod,TAB,IIF(ROW())>=WROW()-1.AND.INKEY(0)#0,',','')
                     ENDSCAN
                     ENDIF
                     SELECT a
                     ENDSCAN
                     WAIT 'просмотр закончен' WINDOW
                     DEACTIVATE WINDOW perem
                     POP KEY
                     RETURN

PROCEDURE upak      &&-----Процедура упаковки
PUSH KEY CLEAR
SELECT a
SET DELETE OFF      && Доступ к записям, помеченным для удаления
SCAN FOR DELETE()   && Просмотр всех помеченных записей в KADR.DBF
                     SELECT b
                     IF SEEK(a.tab)      && Если в базе PEREM.DBF есть записи
                                             && с совпадающими табельными
                                             && номерами, они также помечаются
                     DELETE WHILE a.tab=b.tab
                     ENDIF
                     SELECT a
                     ENDSCAN
                     PACK                && Упаковка базы KADR.DBF
                     SELECT b
                     PACK                && Упаковка базы PEREM.DBF
                     SET DELETED ON
                     POP KEY
                     RETURN
*-----Конец модуля KADRP.RPG-----

```


Хотя в BROWSE-окне выводятся все поля базы PEREM.DBF, поле TAB показано только для наглядности. На практике, конечно, оно не должно быть доступно для какого-либо изменения пользователем. Однако "родительское" поле TAB в базе KADR.DBF, естественно, остается доступным, и это является потенциально опасным фактором.

Недостатком связи по полю, содержание которого пользователь может изменить, является зависимость правильной работы системы от внимательности оператора. Так, пусть в нашем случае пользователь ввел очередную запись в базу KADR.DBF и все сцепленные с ней по табельному номеру записи — во вспомогательную базу PEREM.DBF, а потом заметил, что сделал в табельном номере ошибку. Следствие исправления такой ошибки в базе KADR.DBF — все сцепленные записи станут недоступными программе, хотя и останутся в базе. Таким образом, любые изменения табельного номера в базе KADR.DBF должны сопровождаться изменениями содержимого полей TAB всех сцепленных записей в базе PEREM.DBF. Вообще это возможно сделать с помощью функции проверки ввода в поле TAB, которая в случае изменения данных "перетрясет" все сцепленные записи вспомогательной базы.

Однако наилучшим решением является сцепление баз не по "живому" полю (у нас TAB), а по некоторому специально созданному для этих целей полю числового типа, которое должно быть и в основной и во вспомогательной базах (поле TAB во вспомогательной базе тогда удаляется). Назовем такое поле, например, NOM. Оно никогда не должно быть доступно пользователю и по смыслу является номером ввода данной записи в базу KADR.DBF. Это не номер записи в базе, который может изменяться при упаковке удаленных записей, а номер по порядку, который для каждой новой записи автоматически увеличивается на единицу и остается уникальным и неизменным, вплоть до удаления записи.

Здесь возникает вопрос о длине такого поля. Выбирая его разрядность, следует учитывать, что номер очередного ввода может быть больше, чем число фактических записей в базе, поскольку в ней возможны удаления. За несколько лет эксплуатации системы при интенсивном обновлении базы эта разница может быть весьма значительной.

Технически построить такую программу так же несложно, как и предыдущую. Ее особенностью является только то, что в начале программы следует установить указатель записей на самую последнюю запись базы, выяснить значение поля NOM (номер последнего ввода) данных и сохранить его, например в переменной N. После этого в новую запись базы KADR.DBF в поле NOM следует поместить значение $N=N+1$, которое пользователь, конечно, не должен вычислять и вводить сам. Делаться это должно автоматически при каждом дополнении базы KADR.DBF. В виду этого, теперь придется отказаться от дополнения базы простым нажатием клавиш Ctrl-N. Потребуется написать собственную процедуру дополнения (можно для тех же клавиш).

Сейчас рассмотрим более сложный пример четырехуровневой системы с древовидной организацией данных.

Это справочная система, где на первом уровне представлены данные о предприятиях (база PRED.DBF), на втором — данные о цехах (база СЕН.DBF), на третьем — об участках (UCH.DBF), на четвертом — о бригадах (BRIG.DBF). Сами данные будут очень простыми — только название предприятия, цеха, участка, бригады (поля NAZ типа C длиной 30 символов), поскольку сейчас нас интересует лишь вопрос организации связи между базами. Она будет осуществляться с помощью специальных кодовых полей (KOD) и номеров объектов (поле NOM). На рис.22.8 показаны эти связи,

структуры баз, занимаемые ими рабочие области и использованные команды индексирования.

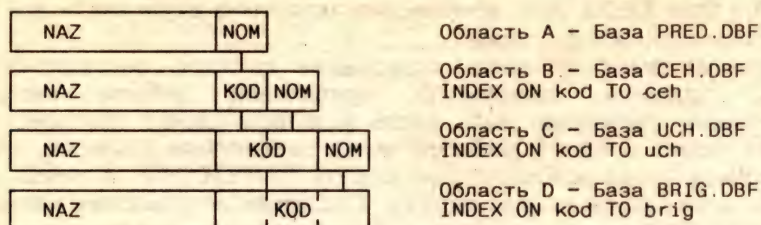


Рис.22.8

Будем считать, что предприятий, цехов на предприятии, участков в цеху не может быть больше 99 и отведем под них символьные поля NOM длиной два разряда. Поле KOD каждой базы данных будет соединением всех полей NOM баз более высоких уровней:

$B.KOD=A.NOM$, $C.KOD=B.KOD+B.NOM$, $D.KOD=C.KOD+C.NOM$

Поэтому под поле B.KOD отведем два разряда, под C.KOD — четыре, под D.KOD — шесть. Все поля символьного типа.

Начальный вариант программы (PRED.PRГ) приведен ниже.

*-Программа PRED.PRГ--нужны файлы PRED.DBF, CEH.DBF, UCH.DBF, BRIG.DBF
 *-----CEH.IDX, UCH.IDX, BRIG.IDX

```

SET SAFETY OFF
SET ESCAPE OFF
SET TALK OFF
SET CARRY OFF
CLEAR MACRO
ON KEY
USE pred IN a          && Открытие баз данных
USE ceh IN b INDEX ceh
USE uch IN c INDEX uch
USE brig IN d INDEX brig
ON KEY LABEL f3 DO ceh  && Назначение на клавишу F3 процедуры CEH
SELECT a
BROWSE                  && Предъявление базы PRED.DBF

PROCEDURE ceh           &&-----Процедура CEH
SELECT b
SET CARRY TO kod        && Режим копирования поля KOD
IF !SEEK(a.nom)         && Если цехов с таким кодом нет,
  APPEND BLANK          && добавляется новая запись в CEH.DBF,
  REPLACE kod WITH a.nom && в которую переносится поле A.NOM
ENDIF
ON KEY LABEL f3 DO uch  && Назначение на F3 процедуры UCH
BROWSE KEY a.nom        && Предъявление записей из CEH.DBF
                        && с полем B.KOD, равным полю A.NOM
ON KEY LABEL f3 DO ceh  && Восстановление старых установок
SELECT a
RETURN

PROCEDURE uch           &&-----Процедура UCH
SELECT c
IF !SEEK(b.kod+b.nom)   && Если участков с таким кодом нет,
  APPEND BLANK          && в UCH.DBF добавляется новая запись,
                        && куда переносятся поля B.KOD и B.NOM
  REPLACE kod WITH b.kod+b.nom
ENDIF
SET CARRY TO kod
ON KEY LABEL f3 DO brig && Назначение на F3 процедуры BRIG
BROWSE KEY b.kod+b.nom  && Предъявление записей из UCH.DBF с
                        && полем KOD равными полям B.KOD и B.NOM
ON KEY LABEL f3 DO uch
SELECT b
  
```



```
RETURN

PROCEDURE brig      &&-----Процедура BRIG
SELECT d
IF !SEEK(c.kod+c.nom)      && Если бригад с таким кодом нет,
  APPEND BLANK              && в BRIG.DBF добавляется новая запись,
                           && куда переносятся поля C.KOD и C.NOM
  REPLACE kod WITH c.kod+c.nom
ENDIF
SET CARRY TO kod
ON KEY                  && Отмена назначений на клавиши
BROWSE KEY c.kod+c.nom  && Предъявление записей из базы BRIG.DBF
                           && с полем KOD, равным полям C.KOD и C.NOM

ON KEY LABEL f3 DO brig
SELECT c
RETURN
*-----Конец модуля PRED.PRG-----
```

База PRED.DBF обрабатывается в основной программе, база СЕН.DBF — в процедуре СЕН, UCH.DBF — в UCH, BRIG.DBF — в BRIG. Для перемещения по базам данных и процедурам используется клавиша F3 (обратное движение — при нажатии клавиши Escape).

Во всех базах, кроме BRIG.DBF, в поле NOM мы вводим номера предприятий/цехов/участков в виде последовательного числового ряда ('01', '02',...). Именно эти номера и определяют принадлежность к данной записи всех записей из следующей базы.

При входе в каждую базу нижнего уровня сначала выполняется команда вида IF !SEEK(<область>.nom), по которой в ней ищутся записи, где кодовое поле KOD совпадает с номером из предыдущей базы. Если таких нет, то база дополняется одной записью, в которой кодовое поле делается равным номеру из предыдущей базы. Собственно доступ к нужному множеству записей, принадлежащих только этому предприятию/цеху/участку, выполняется командами BROWSE с параметром KEY, где в качестве ключа фигурирует номер из предыдущей базы.

При дополнении базы с использованием клавиш Ctrl-N, кодовое поле только повторяется. Этот режим определяет команда SET CARRY TO KOD.

Недостатком программы является необходимость ручного ввода номеров объектов (полей NOM). Делать это необходимо очень внимательно, не допуская ни в коем случае их повторения. Как уже говорилось, желательно вообще освободить пользователя от ввода такой технической информации и даже от ее отображения на экране.

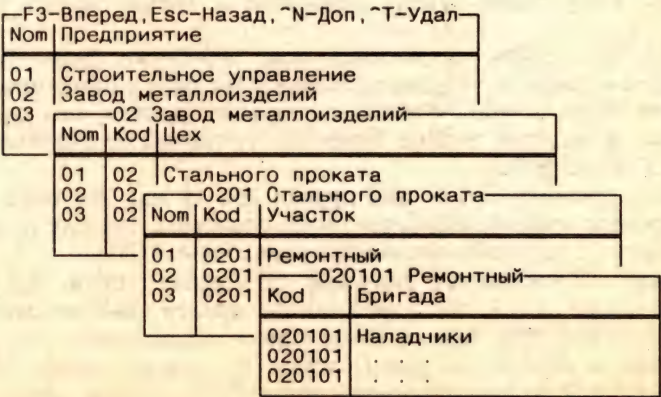


Рис.22.9

В следующей программе (PRED1.PRG) этот вопрос решен. Кроме того, здесь предусмотрен некоторый дизайн, для чего определено четыре окна (PRED, СЕН, UCH, BRIG).

На рис.22.9 отображен экран компьютера в момент, когда пройден весь путь по дереву и вызвана база самого нижнего уровня BRIG.DBF.

В заголовке первого окна PRED указаны управляющие клавиши. В заголовке каждого следующего — код и номер "старшего" объекта и его название. Это дает возможность пользователю видеть весь пройденный им по дереву путь, чтобы не забыть, например, какому предприятию, цеху, участку принадлежит данная бригада.

Поля NOM и KOD отображаются в окнах BROWSE только для наглядности. В готовой программе они должны быть из окна изъяты.

Программа PRED1.PRG приведена ниже. Главное отличие ее от предыдущей заключается в том, что пользователь уже не должен сам вводить номера объектов. В виду этого здесь пришлось отказаться от автоматического дополнения базы клавишами Ctrl-N и написать процедуру для этих клавиш — CTRLN. Разберем ее.

В процедуре обрабатываются три ситуации в зависимости от того, какая база дополняется.

Если это база PRED.DBF из рабочей области 1 (Select(=1), то указатель записей ставится на самую последнюю ее запись, из нее выбирается значение поля NOM и увеличивается на единицу. Результат сохраняется в переменной N. В виду того, что поле NOM имеет символьный тип, такое действие требует нескольких трансформаций. Сначала поле преобразуется в числовой тип и к нему прибавляется единица (VAL(NOM)+1), затем снова приводится к строковому типу и удаляются ведущие пробелы (LTRIM(STR(...))). Далее к полученному выражению слева дописывается символьный "0", и из него выбираются два разряда справа. Ноль следует дописывать в виду того, что новый номер может быть как одно-, так и двухразрядной строкой. В первом случае ноль нужен как часть строки, во втором он не нужен и функцией RIGHT(...) будет отброшен. Пусть, например, значение поля NOM равно соответственно "05" и "12". Тогда для них будут выполнены следующие цепочки преобразований:

```
'04' -> 4 -> 5 -> '5' -> '5' -> '05' -> '05'
'12' -> 12 -> 13 -> '13' -> '13' -> '013' -> '13'
```

После этого полученная строка заносится в новую запись базы в поле NOM.

Если вызов процедуры произошел при обработке базы BRIG.DBF из рабочей области 4 (Select(=4), то, поскольку это база самого нижнего уровня, здесь нет поля NOM и единственно, что нужно сделать, это дополнить базу новой записью, в которой только будет продублировано значение поля KOD (оно равно C.KOD+C.NOM).

При нажатии клавиш Ctrl-N для других баз (у нас это базы СЕН.DBF и UCH.DBF) процесс дополнения будет более сложным, так как нужно отыскать последнюю запись с текущим значением кода (поле KOD).

Сначала здесь запоминается значение ее кодового поля (KK). Затем для того чтобы достичь самой последней такой записи, все множество записей (SCAN WHILE KK=KOD) базы с этим кодом сканируется. Поскольку после выполнения цикла SCAN мы оказываемся на записи, сразу следующей за последней, имеющей нужное значение поля KOD, нужно вернуться на одну строку назад (SKIP -1).

Наконец, добавляется одна запись, поле NOM которой увеличивается на

единицу (переменная N), и дублируется поле KOD (переменная KK).

```
*---Программа PRED1.PRG---нужны файлы PRED.DBF, CEH.DBF,-----
*----- UCH.DBF, BRIG.DBF и CEH.IDX, UCH.IDX, BRIG.IDX -----
CLEAR
SET ESCAPE OFF
SET TALK OFF
SET SAFETY OFF
SET CARRY OFF
CLEAR MACRO
DEFINE WINDOW pred FROM 0,0 TO 12,37      && Описание окон
DEFINE WINDOW ceh FROM 5,4 TO 16,45
DEFINE WINDOW uch FROM 9,12 TO 20,51
DEFINE WINDOW brig FROM 13,22 TO 27,62
USE pred IN a                               && Открытие баз
USE ceh IN b INDEX ceh
USE uch IN c INDEX uch
USE brig IN d INDEX brig
SELECT a
ON KEY LABEL f3 DO ceh
ON KEY LABEL ctrl+n DO ctrln               && Назначение на клавишу ~N
                                           && процедуры дополнения базы
IF EOF()                                   && Если база PRED.DBF пуста,
    APPEND BLANK                          && добавляется новая запись, где в поле
    REPLACE a.nom WITH '01'               && NOM заносится начальный код '01'
ENDIF
                                           && Предъявление базы PRED.DBF
BROWSE TITLE 'F3-Вперед, Esc-Назад, ~N-Доп, ~T-Удал';
COLOR SCHEME 10 WINDOW pred;
FIELD nom :R;;
      naz :H='Предприятие'

PROCEDURE ceh      &&-----Процедура CEH
SELECT b
IF !SEEK(a.nom)    && Если цехов с таким кодом нет,
    APPEND BLANK   && добавляется новая запись в CEH.DBF;
                  && в поле B.KOD которой заносится A.NOM , а в B.NOM - '01'
    REPLACE kod WITH a.nom, nom WITH '01'
ENDIF
ON KEY LABEL f3 DO uch
BROWSE TITLE a.nom+' '+a.naz WINDOW ceh COLOR SCHEM 10 KEY a.nom;
FIELD nom :R;;
      kod :R;;
      naz :H='Цех'
ON KEY LABEL f3 DO ceh
SELECT a
RETURN

PROCEDURE uch      &&-----Процедура UCH
SELECT c
IF !SEEK(b.kod+b.nom) && Если участков с таким кодом нет,
    APPEND BLANK   && добавляется новая запись в UCH.DBF
                  && в которую копируются коды
    REPLACE kod WITH b.kod+b.nom, nom WITH '01'
ENDIF
ON KEY LABEL f3 DO brig
BROWSE TITLE b.kod+b.nom+' '+b.naz WINDOW uch COLOR SCHEME 10;
KEY b.kod+b.nom;
FIELD nom :R;;
      kod :R;;
      naz :H='Участок'
ON KEY LABEL f3 DO uch      && Восстановление старых установок
SELECT b
RETURN

PROCEDURE brig      &&-----Процедура BRIG
SELECT d
IF !SEEK(c.kod+c.nom)
    APPEND BLANK
    REPLACE kod WITH c.kod+c.nom
ENDIF
ON KEY LABEL f3
BROWSE TITLE c.kod+c.nom+' '+c.naz WINDOW brig COLOR SCHEME 10;
KEY c.kod+c.nom;
FIELD kod :R;;
```



```

      naz :H='Бригада'
ON KEY LABEL f3 DO brig
SELECT c
RETURN

PROCEDURE ctrln      &&-----Процедура дополнения CTRLN
DO CASE
  CASE SELECT( )=1      && Дополнение базы PRED.DBF
    GO BOTTOM
    n=RIGHT('0'+LTRIM(STR((VAL(nom)+1),2)),2)      && Вычисление нового кода
    APPEND BLANK
    REPLACE nom WITH n
  CASE SELECT( )=4      && Дополнение базы BRIG.DBF
    APPEND BLANK
    REPLACE kod WITH c.kod+c.nom
  OTHERWISE      && Дополнение остальных баз
    kk=kod      && Сохранение кода
    SCAN WHILE !k=kod      &&- Проход всех записей с этим кодом
    ENDSCAN
    SKIP -1      && Возвращение на одну запись назад
    n=RIGHT('0'+LTRIM(STR((VAL(nom)+1),2)),2)
    APPEND BLANK      && Дополнение записи
    REPLACE nom WITH n, kod WITH kk      && Занесение нового номера и прежнего кода
ENDCASE
RETURN
*-----Конец модуля PRED1.PRG-----

```

Возможна и другая техника организации древовидной структуры данных — с помощью команд SET RELATION. Ниже приведена часть программы PRED2.PRG, в которой предьявляются, но уже одновременно те же самые четыре связанные базы данных.

Сначала в программе определяются окна для каждой из баз. Далее производятся назначения клавиш. Нажатие клавиш Ctrl-N вызывает дополнение текущей базы новой записью (процедура DOP), F8 — обработку сделанного выбора из всех баз (процедура F8), Ctrl-End/Esc — завершение программы (KON). Клавиши F1 — F4 задействованы для перемещения между одноименными окнами.

Как ранее, указывалось BROWSE-окно, имеющее опцию TITLE, или помещенное в другое окно с этой опцией, далее в качестве имени считает не имя окна, а этот заголовок (латинские буквы и цифры до первого пробела или небуквенного или нецифрового символа). Поскольку нам придется программным образом переключать/отключать окна, например нажатием клавиш F1-F4, именно эти символы мы поместим в начало каждого TITLE-заголовка в командах BROWSE. За данными клавишами мы закрепим действия по перемещению между окнами, так что клавиша F1 будет вызывать перемещение курсора в окно базы PRED.DBF, F2 — в окно базы СЕН.DBF и т.д. Несмотря на то, что перемещение между окнами может быть осуществлено с помощью мыши и клавиш Ctrl-F1, такие назначения будут весьма полезными.

Затем открываются базы и вызывается процедура BAZA для их предьявления. Здесь сначала устанавливается связь базы PRED.DBF с базой СЕН.DBF, затем СЕН.DBF с UCH.DBF, и наконец UCH.DBF с BRIG.DBF, а также отображается клавишное меню в верхней части экрана.

Далее все базы предьявляются от старшей к младшей. Для того чтобы активной в начальный момент была главная база PRED, она активируется специальной командой ACTIVATE WINDOW f1 (F1 теперь имя окна этой базы). Поскольку при перемещении между окнами BROWSE (с помощью клавиш F1-F4, Ctrl-F1 или мыши) выделена цветом текущая запись только активной в данный момент базы, справки об этих записях для всех баз выводятся в 24-й строке экрана функцией SPR(). Из окна BROWSE в

функцию передаются номера колонок вывода и объекты вывода (в данном случае — коды и номера предприятий, цехов и т.д.). В готовой программе нужно выводить, конечно, не служебные реквизиты записей, а их содержимое — названия предприятий, цехов. Для того чтобы можно было работать сразу с несколькими базами одновременно, команды BROWSE используют опции NOWAIT. Ниже всех команд BROWSE поставлена команда READ VALID kkk (где kkk=F.). Это предотвращает завершение программы после предъявления последнего окна BROWSE. Все такие окна остаются "подвешенными" до явного указания выхода из программы с помощью клавиш Esc/Ctrl-End, нажатие на которые вызывает процедуру окночания работы KON. В этой процедуре по команде RELEASE удаляются все BROWSE-окна, а переменная kkk присваивается значение .T., позволяющее далее выйти из команды READ.

Процедура DOP реализует дополнение активной базы новой записью при нажатии клавиш Ctrl-N. Здесь сначала выясняется (K) номер рабочей области, из которой произошел запрос на дополнение, а также значение функции конца файла (E).

Далее в зависимости от того, какая база дополняется, после команды APPEND BLANK в поле KOD новой записи копируются ключевые (для распознавания принадлежности данной записи к записи из старшей базы) поля. Чтобы не допустить дополнения младшей базы новой записью, в то время как в старшей еще нет соответствующей записи, контролируется значение поля NOM в старшей базе. Если оно пусто (EMPTY(nom)), дополнения не происходит. При этом выдается соответствующее сообщение. В завершение проверяется значение функции EOF(), полученной в начале процедуры. Если E=.T., значит указатель стоял в конце базы. Это возможно в двух случаях. Во-первых, если база пуста, во-вторых, если она не пуста, но в ней нет ни одной связанной записи. Последнее возможно только для младших баз. В обоих случаях, хотя дополнение базы и произойдет, оно не будет отражено в окне без обновления его изображения, например по команде SHOW WINDOW (WONTOP()) REFRESH. Аргументом команды является имя активного в данный момент окна. Только после этого новая запись появится в окне. Однако имеется одна особенность (недоработка системы) — курсор отображен не будет. Чтобы он появился, нужно перебросить курсор в другое окно (например, в окно F1), а затем вернуться обратно. Эти действия в программе осуществляются по командам ACTIVATE WINDOW f1 и ACTIVATE WINDOW (x), где X — имя текущего окна.

При нажатии клавиши F8 вызывается одноименная процедура, где предъявляются отобранные записи из всех баз и запрашивается решение об их обработке. Если решение положительное (нажато Enter), вызывается процедура OTBOR. Эта процедура, а также другие функции системы здесь не рассматриваются.

```
*-----Фрагмент программы PRED2.PRG-----
ON KEY LABEL Ctrl+N DO dop
ON KEY LABEL f8 DO f8
ON KEY LABEL Ctrl+End DO kon
ON KEY LABEL Esc DO kon
ON KEY LABEL f1 ACTIVATE WINDOW f1
ON KEY LABEL f2 ACTIVATE WINDOW f2
ON KEY LABEL f3 ACTIVATE WINDOW f3
ON KEY LABEL f4 ACTIVATE WINDOW f4
DEFINE WINDOW pred FROM 1,1 TO 14,20 COLOR SCHEME 10 && Окна
DEFINE WINDOW ceh FROM 3,21 TO 17,40 COLOR SCHEME 10
DEFINE WINDOW uch FROM 5,41 TO 20,60 COLOR SCHEME 10
DEFINE WINDOW brig FROM 7,61 TO 23,79 COLOR SCHEME 10
DEFINE WINDOW otbor FROM 5,5 TO 12,70 COLOR SCHEME 5;
TITLE 'ОТБОРАНО'
USE pred IN a
```

```
&& Назначение клавиш:
&& Дополнение
&& F8 - Отбор
&& ^End - конец
&& Esc - конец
&& Переключение окон
```

```
&& Открытие баз
```



```

USE ceh IN b INDEX ceh
USE uch IN c INDEX uch
USE brig IN d INDEX brig
DO baza          && Вызов баз данных, конец головной части программы

PROCEDURE baza    &&-----Процедура предъявления связанных баз
SELECT a          && Установление связей
SET RELATION TO a.nom INTO b
SELECT b
SET RELATION TO b.kod+b.nom INTO c
SELECT c
SET RELATION TO c.kod+c.nom INTO d
SELECT d
@0,25 TO 2,69 DOUBL          && Строка-меню
@1,27 SAY 'F8 - ОТБОР      Esc/~End - КОНЕЦ      'N - ДОП.' COLOR r/w
SELECT a              && Открытие окон BROWSE
BROWSE WINDOW pred NOWAIT;
  FIELD nom :H='NN' :P='99' :V=!EMPTY(nom) :F,;
  naz :H='Название';
  WHEN spr(0,nom) TITLE 'F1 Предприятие'
SELECT b
BROWSE WINDOW ceh NOWAIT;
  FIELD kod :H='Код' ;
  nom :H='NN' :P='99' :V=!EMPTY(nom) :F,;
  naz :H='Название';
  WHEN spr(20,kod+nom) TITLE 'F2 Цех'
SELECT c
BROWSE WINDOW uch NOWAIT;
  FIELD kod :H='Код' ;
  nom :H='NN' :P='99' :V=!EMPTY(nom) :F,;
  naz :H='Название';
  WHEN spr(40,kod+nom) TITLE 'F3 Участок'
SELECT d
BROWSE WINDOW brig NOWAIT;
  FIELD kod :H='Код' ;
  naz :H='Название';
  WHEN spr(60,kod) TITLE 'F4 Бригада'
ACTIVATE WINDOW f1          && Переход в базу Предприятия
kkk=.F.                    && Переменная выхода из READ (Запрещен)
READ VALID kkk
RETURN

PROCEDURE kon        &&-----Процедура завершения
RELEASE WINDOWS f1,f2,f3,f4          && Удаляются все BROWSE-окна
CLEAR READ          && Завершение READ
kkk=.t.            && Выход разрешен
RETURN

FUNCTION spr          &&-----Функция вывода на экран текущих записей
PARAMETERS j,x
@ 24,j SAY x COLOR w+/r
RETURN

PROCEDURE f8          &&-----Процедура вывода отобранных данных
PUSH KEY CLEAR
ACTIVATE WINDOW otbor
@ 0,1 SAY 'Предприятие - ' +a.nom+ ' ' +a.naz
@ 1,1 SAY 'Цех - ' +b.nom+ ' ' +b.naz
@ 2,1 SAY 'Участок - ' +c.nom+ ' ' +c.naz
@ 3,1 SAY 'Бригада - ' +d.naz
@ 4,0 TO 4,63
@ 5,10 SAY 'Нажать Enter для отбора (отказ - любая клавиша)'
IF INKEY(0)=13          && Если нажата Enter,
  DO otbor              && вызов процедуры обработки выбора
  WAIT 'Отобрано' WINDOW
ENDIF
DEACTIVATE WINDOW otbor
POP KEY
RETURN

PROCEDURE dop        &&-----Процедура дополнения баз
k=SELECT()          && Номер области, где находится база
e=EOF()             && Значение функции "Конец файла"
DO CASE
  CASE k=1          && Область 1 (Предприятия)

```



```

APPEND BLANK
CASE k=2.AND.!EMPTY(a.nom)      && Область 2 (Цеха)
APPEND BLANK
REPLACE kod WITH a.nom
CASE k=3.AND.!EMPTY(b.nom)      && Область 3 (Участки)
APPEND BLANK
REPLACE kod WITH a.nom+b.nom
CASE k=4.AND.!EMPTY(c.nom)      && Область 4 (Бригады)
APPEND BLANK
REPLACE kod WITH a.nom+b.nom+c.nom
OTHERWISE      && Если в записи из старшей области нет Номера
WAIT 'Дополнение невозможно' WINDOW && выдается сообщение
ENDCASE
IF e      && Если ранее такого подмножества не было,
x=WONTOP()      && запоминается имя текущего окна,
SHOW WINDOW (x) REFRESH      && окно обновляется,
ACTIVATE WINDOW f1      && временно активируется другое окно,
ACTIVATE WINDOW (x)      && вновь активируется текущее окно
ENDIF
RETURN
*-----Конец модуля PRED2.PRG-----

```

Предложенная программа по сравнению с предыдущей реализует более удобный интерфейс пользователя. Здесь он может одновременно видеть и быстро переходить из окна в окно. Перемещение курсора в старшем окне автоматически изменяет содержание всех подчиненных окон. Вместе с тем, если вы из младшего окна перешли в старшее, а затем вернулись назад, курсор станет не на покинутую, а на первую запись младшего окна. Кроме того, одновременное перемещение указателей записей сразу в нескольких базах данных требует определенного времени.

До сих пор мы к каждой записи главной базы подбирали записи из младших баз. Решим теперь обратную задачу — розыск для данных из младшей базы соответствующих записей старшей. Применительно к предыдущей задаче это был бы, например, розыск для каждого цеха соответствующего предприятия.

Сейчас мы разберем следующую задачу. Пусть имеется база предприятий PREDPR.DBF, содержащая два поля: поле названий предприятий NAZ и поле, содержащее список, перечисленных через запятую, наименований видов, производимой ими продукции PROD. Это поле может иметь значительную длину и даже быть мемо-полем. Цель заключается в том, что бы построить систему отбора записей из базы PREDPR.DBF по наименованию нужного изделия. Тривиальное решение может быть получено, например, с помощью команды задания фильтра вида

SET FILTER TO <название изделия> \$ PROD

или команд с аналогичным FOR-условием. Однако отбор данных таким образом будет выполняться недопустимо медленно. Если база PREDPR.DBF имеет значительный размер, время отклика системы составит минуты. Кроме того, неплохо иметь меню, содержащее перечень всех встречающихся изделий.

Предлагаемое решение позволяет организовать быстрый доступ к нужным данным. Но для этого придется выполнить сложный процесс анализа основной и создания вспомогательной базы данных. Алгоритм поясняет рис. 22.10.

1. Сначала из основной базы PREDPR.DBF создается база IZD.DBF, содержащая в поле IZD наименования всех видов изделий (по одному в поле), а в поле REC — физические номера записей базы PREDPR.DBF, в которых встретились эти наименования.

2. Следующий этап — сортировка базы IZD.DBF по полям IZD (главное поле сортировки) и REC (вторичное поле). Новая база имеет имя IZDSORT.DBF. После этого ненужная более база IZD.DBF может быть

очищена (ZAP) и закрыта. Данные в базе IZDSORT.DBF располагаются в алфавитном порядке названий изделий причем для повторяющихся названий, первой идет запись с наименьшим значением поля REC.

3. База IZDSORT.DBF индексируется по полю IZD с опцией UNIQUE. Это позволит нам в дальнейшем создать POPUP-меню из полей IZD, в котором все наименования изделий встречаются только по разу.

PREDPR.DBF			→		IZD.DBF		→		IZDSORT.DBF	
NN записи	NAZ		PROD		REC	IZD	REC	IZD		
18	з-д Алмаз	сверла, фрезы			18	сверла	561	насосы		
561	з-д Ротор	фрезы, насосы			18	фрезы	940	насосы		
940	п/о Сокол	станки, насосы			561	фрезы	18	сверла		
					561	насосы	940	станки		
					940	станки	18	фрезы		
					940	насосы	561	фрезы		

Рис.22.10

Естественно, что такой процесс не может быть быстрым. Однако, создание базы IZDSORT.DBF выполняется уже после того, как база PREDPR.DBF заполнена и обновление ее делается редко. Если после этого в ней произошли какие-то изменения, они, естественно, не отразятся автоматически в базе IZDSORT.DBF и всю процедуру нужно запускать снова. Вместе с тем после создания базы IZDSORT.DBF доступ к нужным записям в базе PREDPR.DBF будет выполняться очень быстро. Поскольку известны номера всех этих записей, их не нужно искать. Кроме того, в программе реализован процесс автоматического создания меню выбора непосредственно из имеющихся данных.

Очевидно также, что индексирование по полю PROD не имеет смысла, так как оно в общем случае содержит сразу несколько наименований. Программа, IZD.PRГ реализующая приведенный алгоритм, изображена ниже.

База PREDPR.DBF открывается в области А, база IZDSORT.DBF - в области В совместно с индексным файлом IZD.IDX. В области D открывается база TEMP.DBF, в которую в дальнейшем производится отбор групп предприятий, производящих нужные виды продукции. Поскольку это временная и сравнительно небольшая база, ее удобно разместить на более быстром виртуальном диске Е. С этой целью по команде COPY файл TEMP.DBF копируется из текущей директории на диск Е. Если такого диска на компьютере нет, команда COPY вызовет ошибку (кодшибки — 202) и база TEMP.DBF будет открыта в текущей директории обычного диска в области D, поскольку в случае ошибки по команде ON ERROR будет сделано присвоение переменной F значения "temp" (без указания диска). Если ошибки не произошло, т.е. диск Е существует, F останется равной "e:temp".

Далее определяются окна: окно PRED — для вывода основной базы PREDPR.DBF и окно TEMP для предприятий, выпускающих желаемую продукцию. Затем определяется VAR-меню вида

Предприятия	Изделия	Отобрано
-------------	---------	----------

где пункт "Предприятия" предъявляет все предприятия — базу PREDPR.DBF (процедура PRED), пункт "Изделия" реализует алгоритм создания вспомогательной базы IZDSORT.DBF (процедура KLUCH), а пункт

"Отобрано" вызывает редактор для просмотра текстового файла с конкретными, отобранными пользователем предприятиями (процедура PROSM). Кроме того, здесь определяется POPUP-меню IZD, образованное из полей IZD базы IZDSORT.DBF. Затем идут процедуры.

Процедура KLUCH (главная для нас) формирует базу IZDSORT со всеми ключевыми словами (наименованиями изделий) из поля PROD базы PREDPR.DBF. Для этого сканируются все записи с непустыми полями PROD, из которых отбираются отдельные названия изделий. Названием считается любое слово/слова до запятой или до конца поля. Считается, что предприятие может выпускать до 50 различных наименований изделий (FOR i=1 TO 50). После формирования базы IZD.DBF она сортируется по полям IZD и REC с занесением результатов в базу IZDSORT.DBF, которая, в свою очередь, индексируется по полю IZD с опцией UNIQUE.

В процедуре PRED формируется BROWSE-окно базы PREDPR.DBF и делаются закрепления за клавишами: Ctrl-Enter — процедуры выбора предприятий (VIBOR), F2 — вызова POPUP-меню изделий IZD.

Обработка выбора в POPUP-меню реализуется в процедуре OTBOR, которая все записи предприятий, производящих выбранный в меню вид продукции, пересылает во временную базу TEMP для просмотра и дальнейшего отбора. С этой целью в базе IZDSORT.DBF отключается главный индекс, что позволяет получить доступ ко всем записям этой базы, имеющим одинаковое, выбранное в меню, значение поля IZD (команда SCAN REST WHILE PROMPT(=b.izd). В каждой записи базы IZDSORT.DBF содержится номер записи базы PREDPR.DBF, где находится предприятие-изготовитель, что позволяет легко установить на нее указатель записей (GO b.rec IN a). Такое поле из PREDPR.DBF пересылается в TEMP.DBF. По завершении формирования базы TEMP.DBF главный индекс восстанавливается (нам еще понадобится меню) и открывается BROWSE-окно базы TEMP.DBF.

Процедура VIBOR реализует отбор пользователем уже конкретных предприятий из множества производителей нужной продукции, как из основной базы PREDPR.DBF, так из базы TEMP.DBF. Отбор выполняется при нажатии клавиш Ctrl-Enter при работе в любом BROWSE-окне. Выбранные записи (предприятия) пересылаются уже в текстовой форме, в текстовый альтернативный файл PREDVIB.TXT. Если такой файл ранее уже был создан, у пользователя один раз запрашивается решение о его судьбе: начать его заново (клавиша Enter) или продолжить дополнение (клавиша Space). В первом случае старый файл удаляется (ERASE). Чтобы запрос не повторялся каждый раз, вводится переменная X, фиксирующая этот факт.

К процедуре PROSM доступ возможен через центральное меню программы (пункт "Просмотр"). Здесь выясняется, открыт ли альтернативный файл (его имя запоминается в переменной S). Это необходимо для того, чтобы в последствии открыть его снова, поскольку редактор может принять только закрытые файлы. Далее выясняется, есть ли вообще этот файл на диске. Если он еще не был создан, выдается соответствующее сообщение и процедура завершается. Если файл есть, вызывается редактор для его просмотра. По завершении просмотра альтернативный файл снова открывается (если перед этим он был открыт).

```
*-----Программа IZD.PRG-----
*--нужны файлы: PREDPR.DBF, IZD.DBF, IZDSORT.DBF, IZD.IDX, TEMP.DBF
SET CONFIRM ON
SET SAFETY OFF
x=.f.      && Признак открытия альтернативного текстового файла
USE predpr IN a      && Основная база предприятий
USE izdsort IN b INDEX izd      && База-меню изделий
f='e:temp'
ON ERROR f='temp'      && Реакция на ошибку копирования
```



```

COPY FILE temp.dbf TO e:temp.dbf.
USE (f) IN d      && Открытие временной базы отобранных записей
ON ERROR
&& Основное окно PRED и окно отобранной группы изделий TEMP
DEFINE WINDOW pred FROM 1,1 TO 15,60 COLOR SCHEME 10
DEFINE WINDOW temp FROM 2,2 TO 16,61 COLOR SCHEME 10
DEFINE MENU mn      && Главное меню
DEFINE PAD mn1 OF mn PROMPT "Предприятия";
MESSAGE 'Предприятия - производители продукции'
DEFINE PAD mn2 OF mn PROMPT "Изделия";
MESSAGE 'Формирование списка видов продукции'
DEFINE PAD mn3 OF mn PROMPT "Отобрано" ;
MESSAGE 'Просмотр выбранных предприятий-поставщиков'
ON SELECTION PAD mn1 OF mn DO pred
ON SELECTION PAD mn2 OF mn DO kluch
ON SELECTION PAD mn3 OF mn DO prosm
DEFINE POPUP izd FROM 2,56 PROMPT FIELD b.izd;
TITLE 'Изделия'      && Меню изделий
ON SELECTION POPUP izd DO otbor
ACTIVATE MENU mn
CLOSE ALTERNATE

PROCEDURE kluch      &&-Процедура создания базы из названий изделий
WAIT "Ждите - идет формирование списка изделий" WINDOW NOWAIT
SELECT b      && Закрытие базы IZDSORT.DBF
USE
USE izd IN c      && Открытие промежуточной базы IZD.DBF
SELECT c
ZAP      && и ее очистка
SELECT a      && Переход в область PREDPR.DBF
GO TOP      && Начало базы
&& Сканирование записей с непустыми полями PROD
SCAN FOR !EMPTY(prod)
n1=1      && Начальная позиция просмотра поля PROD
l=LEN(prod)      && Длина поля PROD
FOR i=1 TO 50      && Сканирование поля PROD
n2=AT(' ',prod,i)      && Положение очередной запятой
&& Выделение наименования одного вида изделия из поля PROD
a=ALLTRIM(SUBSTR(prod,n1,IF(n2>n1,n2-n1,1)))
SELECT c      && Переход в область IZD.DBF
APPEND BLANK      && Дополнение IZD.DBF пустой записью
&& Занесение изделия и номера записи из PREDPR.DBF в IZD.DBF
REPLACE c.izd WITH a,c.rec WITH RECNO(1)
SELECT a      && Возврат в область PREDPR.DBF
IF n2=0      && Если запятой больше нет,
EXIT      && значит, все изделия из поля PROD выбраны
ENDIF
n1=n2+1      && Позиция названия нового изделия после запятой
ENDFOR
ENDSCAN
SELECT c
SORT TO izdsort ON izd,rec && Сортировка данных в базу IZDSORT
ZAP      && Очистка промежуточной базы IZD.DBF
USE      && и ее закрытие
SELECT b      && Открытие базы IZDSORT в области B
USE izdsort
INDEX ON izd TO izd UNIQUE COMPACT && и ее индексирование
WAIT CLEAR
RETURN

PROCEDURE pred      &&-----Процедура доступа к базе предприятий
SELECT a
ON KEY LABEL ctrl+enter DO vibor
ON KEY LABEL f2 ACTIVATE POPUP izd      && F2 вызывает меню
BROWSE FIELD naz :h='Предприятие' ;
prod :h='Продукция';
TITLE 'F2' вызов меню изделий (Выбор - 'Enter') WINDOW pred
ON KEY
RETURN

PROCEDURE prosm      &&--Процедура просмотра отобранных предприятий
s=SET('ALTERNATE',1)      && Сохраняется имя открытого альтер. файла
SET ALTERNATE TO      && Файл отключается
IF !FILE('PREDVIB.TXT')      && Если такого файла нет,

```



```

WAIT 'Альтернативного файла нет' WINDOW NOWAIT
RETURN                                && выход из процедуры
ENDIF
MODIFY FILE predvib.txt NOEDIT        && Вызов редактора для просмотра
IF !EMPTY(s)                          && Если ранее альтернативный файл был открыт,
SET ALTERNATE TO predvib ADDITIVE    && он открывается снова
ENDIF
RETURN

PROCEDURE vibor                       &&-----Процедура выбора предприятий
PUSH KEY CLEAR
&& Если файл PREDVIB.TXT существует, решается что с ним делать
IF FILE('PREDVIB.TXT') AND !x
WAIT 'Файл отбора существует! Переписать-Enter, Дополнить-Space';
WINDOW
DO CASE
CASE LASTKEY()=13                    && Если нажата Enter, удаляется
ERASE predvib.txt                  && старый альтернативный файл
&& Если нажата Space, действий не выполняется
CASE LASTKEY()=32
OTHERWISE                          && Если нажата другая клавиша,
RETURN                              && выход из процедуры
POP KEY
ENDCASE
ENDIF
x=.t.                                && Значение X изменяется на .T.
SET ALTERNATE TO predvib ADDITIVE    && Создается альтернат. файл
SET CONSOL OFF                      && Отключается вывод на консоль
SET ALTERNATE ON                    && Разрешается вывод в файл
?REPLICATE('-',.65)                && и выводится запись
?'Предприятие -',naz
?'Продукция',prod
SET ALTERNATE OFF                    && Вывод прекращается
SET CONSOL ON                       && Восстанавливается вывод на консоль
POP KEY
RETURN

PROCEDURE otbor                       &&-----Процедура отбора изделия
ON KEY LABEL F2
SELECT d                              && Переход в область временной базы TEMP.DBF
ZAP                                  && Ее очистка
SELECT b                              && Переход в область базы IZDSORT.DBF
SET ORDER TO 0                       && Отмена главного индекса
&& Сканирование записей с одинаковым значением поля IZD
SCAN REST WHILE PROMPT()=b.izd
GO b.rec IN a                        && Перемещение указателя записей в PREDPR.DBF
SELECT d                              && Переход в область базы TEMP.DBF
APPEND BLANK                          && Дополнение пустой записью и копирование
REPLACE d.naz WITH a.naz,d.prod WITH a.prod && из PREDPR.DBF
SELECT b                              && Возврат в область IZDSORT.DBF
ENDSCAN
SET ORDER TO 1                       && Восстановление главного индекса
SELECT d                              && Переход в область TEMP.DBF
&& Вывод группы отобранных данных
BROWSE FIELD naz :h='Предприятие';
prod :h='Продукция';
TITLE PROMPT()+' (Выбор - ^Enter)' WINDOW temp
SELECT a
ON KEY LABEL F2 DO otbor
RETURN
*-----Конец модуля IZD.DBF-----

```

22.3. Работа со словарями

При разработке прикладных систем часто приходится сталкиваться с повторяющимися данными, которые имеют большую длину. С целью экономии дисковой памяти и повышения скорости обработки данных такие данные целесообразно хранить в отдельном коротком файле, где каждой строке данных соответствует некоторый назначаемый ей код. Такой файл обычно называется кодификатором, справочником или словарем. Тогда вместо

соответствующей строки данных в основной рабочий файл включается небольшое поле кода.

В связи с этим возникает задача организации взаимодействия файла-словаря и основного файла данных.

Рассмотрим пример. Пусть необходимо создать и поддерживать базу данных PREDPR.DBF о предприятиях, содержащую следующую информацию:

о название министерства или ведомства, которому принадлежит предприятие (поле KODVED);

о регион (республика, область), где оно расположено (KODREG);

о название самого предприятия (поле NAZ);

о количество работающих на нем людей (поле RAB).

Первые два элемента данных имеют большую длину (до 40 символов). В виду этого в основной базе будем хранить не сами данные, а их числовые коды, код ведомства — в поле KODVED, код региона — в поле KODREG.

Сначала построим систему, где словари хранятся в двух отдельных базах KODVED.DBF, KODREG.DBF с идентичными структурами для хранения кодов (поле KOD) и их содержательных значений (поле SOD). Структуры файлов приведены на рис.22.11.

Файлы KODVED.DBF и KODREG.DBF			Файл PREDPR.DBF		
Имя поля	Тип	Длина	Имя поля	Тип	Длина
KOD	Numeric	2	KODVED	Numeric	2
SOD	Character	40	KODREG	Numeric	2
			NAZ	Character	20
			RAB	Numeric	5

Рис.22.11

Типы и длины всех содержательных полей (SOD, NAZ, RAB) определяются фактическим видом данных. Характеристики полей кодов задаются программистом исходя из количества возможных значений кодов. Так как количество министерств/ведомств и областей в стране не менее 10 и не превышает 99, то возьмем для этих полей числовой тип длиной 2. Возможен и символьный тип, в особенности если у заказчика еще до внедрения ЭВМ существовала иная кодировка этих данных, к которой привыкли пользователи.

Фрагмент программы работы с такими словарями-файлами приведен ниже. Для каждого словаря создаются составленные из полей SOD POPUP-меню, с теми же именами (KODREG, KODVED), что и файлы. Одинаковые названия файлов, полей и меню в дальнейшем позволят легко установить связь между всеми элементами системы. Основной экран программы образован командой BROWSE для главной базы PREDPR.DBF. Здесь для полей KODREG, KODVED по нажатию клавиш F2/F3 вызываются процедуры KOD/SLOV для выбора/редактирования кодов регионов и ведомств. При этом для каждого из полей предъявляются свои меню/экраны. Для обработки выбора из обоих меню не назначается никакая процедура. Команда ON SELECTION POPUP ALL только деактивирует меню. Это значит, что обработка выбора будет выполняться непосредственно в программе, где встретится команда ACTIVATE POPUP.

В процедуре KOD сначала в переменной K запоминается имя поля, из которого был сделан вызов (K=VARREAD()). Далее выясняется, начинается ли оно с букв "KOD". Если это так, значит это одно из полей KODREG или KODVED, за которыми закреплены словари. Поскольку вызов пустого меню-файла, в котором нет записей, бессмыслен, анализируется функция конца файла для нужной базы (EOF(K)). Если записей нет, выдается сообщение "Словарь пуст" с выходом из процедуры. Если есть, то активируется нужное

POPUP-меню. Затем после выбора из меню в соответствующее поле базы PREDPR.DBF заносится выбранный код.

Процедура SLOV заполнения словарей организована похожим образом, но здесь предъявляется не POPUP-меню, а BROWSE-окно нужных баз-словарей.

```
*-----Программа SLOV1.PRG-----
*----- нужны файлы PREDPR.DBF, KODREG.DBF, KODVED.DBF -----
SET TALK OFF
CLEAR
USE predpr IN a
USE kodreg IN b
USE kodved IN c
DEFINE POPUP kodreg FROM 5,20 PROMPT FIELD b.sod
DEFINE POPUP kodved FROM 5,20 PROMPT FIELD c.sod
ON SELECTION POPUP ALL DEACTIVATE POPUP
ON KEY LABEL f2 DO kod
ON KEY LABEL f3 DO slov
SELECT a
BROWSE COLOR SCHEME 10 TITLE 'F2/F3 - вызов/заполнение словаря';
    FIELD kodreg :H='Код региона',;
        kodved :H='Код ведомства',;
        naz :H='Название',;
        rab :H='Работников'

PROCEDURE kod          &&-----Процедура вызова словаря по F2
PUSH KEY CLEAR
k=UPPER(VARREAD())      && Имя поля из которого сделан вызов
IF LEFT(k,3)='KOD'      && Если оно начинается на KOD,
    IF EOF(k)           && проверяется наличие данных
        WAIT 'Словарь пуст' WINDOW && Если нет, сообщение
    ELSE                && Иначе
        ACTIVATE POPUP &k      && активируется нужное POPUP-меню
        IF LASTKEY()#27      && Если не нажата клавиша Escape,
            REPLACE a.&k WITH &k..kod      && выбор переносится в базу
        ENDIF
    ENDIF
ENDIF
POP KEY
RETURN

PROCEDURE slov          &&-----Процедура заполнения словаря по F3
PUSH KEY CLEAR
k=UPPER(VARREAD())
IF LEFT(k,3)='KOD'
    SELECT (k)           && Переход в область с нужным именем
    BROWSE COLOR SCHEME 10 TITLE 'Словарь';
        FIELD kod :H='Код',;
        sod :H='Название'
ENDIF
SELECT a
POP KEY
RETURN
*-----Конец модуля SLOV1.PRG-----
```

Разобранная программа является самым примитивным способом организации словарей. На практике это повлекло бы, возможно, необходимость создания слишком многих файлов.

Вообще файл-словарь может быть один на всю систему обработки данных или их может быть столько, сколько имеется основных баз данных с кодируемыми полями, или столько, сколько кодируемых полей (как в приведенном примере). Решение этого вопроса можно принять только "по месту" с учетом количества словарей, их размера, количества баз данных и требований к скорости доступа к словарям.

Ниже предлагается более компактное решение, где все словари будут храниться в одной базе SLOV.DBF, при этом возможен доступ к разным ее подмножествам. База совпадает по структуре с файлами KODVED.DBF и KODREG.DBF, но имеет еще дополнительное поле POLE символьного типа длиной 10

POLE Character 10

для того, что бы распознать, какому полю (KODREG или KODVED) из основной базы принадлежит каждая запись словаря. Размер такого поля не может превышать 10 — максимально разрешенной длины имени в FoxPro. В нашем случае было бы даже достаточно шести, поскольку его значением здесь должно быть только "KODVED" и "KODREG". В качестве ключевого поля можно взять и более короткий параметр, например номер поля KODREG или KODVED в базе PREDPR.DBF (2 и 1 соответственно), что сократит длину ключевого поля до одного разряда.

Для организации связи между обоими файлами нам понадобятся два индекса к базе SLOV.DBF. Индекс KODVED.IDX по полю KOD с условием POLE='KODVED' необходим для выделения из базы-словаря подмножества записей, относящихся к полю (KODVED) основной базы. Аналогичным образом формируется индекс KODREG.IDX.

Индексы создаются по командам

```
.USE slov
INDEX ON kod TO kodved FOR pole='KODVED' COMPACT
INDEX ON kod TO kodreg FOR pole='KODREG' COMPACT
```

Программа, реализующая нашу задачу, приведена далее. В отличие от предыдущей она имеет некоторый пользовательский интерфейс, включая окна, заголовки, процедуры упаковки, заполнения словарей и вывода данных.

Сначала здесь открываются базы данных и индексы, описываются окна ввода данных в базу PREDPR.DBF и SLOV.DBF. Затем формируется POPUP-меню KOD из поля SOD словаря. Наполнение такого меню на экране будет определяться активным индексом.

Главное двухуровневое меню SLOV изображено на рис.22.12. Курсор стоит в позиции Словари.

Предприятия	Вывод	Словари	Упаковка	Конец
		Словарь ведомств Словарь регионов		

Рис.22.12

Меню активируется командой ACTIVATE MENU SLOV, и из него производится вызов необходимых процедур.

Процедура SLOV1 реализует возможность ввода и редактирования словарей. В нее из меню передаются параметры:

K=BAR() и P=PROMPT() — номер выбранного пункта в POPUP-меню и его содержимое. Последнее нужно для задания заголовка POPUP-меню. Далее осуществляется переход в область В (область файла-словаря SLOV.DBF) и устанавливается главным нужный индекс К (1 или 2). Затем определяется окно SLOV1 для работы со словарем. Если файл-словарь пуст (EOF()), то он дополняется пустой записью, где в поле POLE заносится имя нужного поля (KODVED/KODREG), которое совпадает с именем главного индекса. И наконец, по команде BROWSE разворачивается экран ввода-редактирования словаря. Поле POLE здесь не предьявляется, поскольку дополнение новых записей в базу-словарь вызовет автоматическое занесение в поле POLE имени кодируемого поля (SET CARRY TO POLE) без участия пользователя. Экран словаря показан на рис.22.13.

Процедура PREDPR формирует основное окно ввода данных в базу PREDPR.DBF. В команде BROWSE для полей KODVED и KODREG

установлена функция проверки ввода V=KOD(). Через эту функцию происходит вызов POPUP-меню нужного словаря нажатием клавиши F2 при работе в указанных полях. Для разнообразия вызов процедур выполняется не с помощью команд ON KEY LABEL, как в предыдущем случае, а с помощью команды SET FUNCTION, по которой за клавишей F2 закрепляется символ клавиши выхода из поля Enter (";"). В команде BROWSE такое нажатие распознается и обрабатывается в функции KOD() (опции :V=kod() :F в полях KODREG и KODVED).

Словарь ведомств	
Код	ВЕДОМСТВА
1	Министерство сельского хозяйства
2	Комитет по науке и образованию

Рис.22.13

В функции KOD() сначала проверяется, была ли нажата клавиша F2 (LASTKEY()=1). Если да, то в переменную К заносится имя поля, из которого был сделан вызов, осуществляется переход в область В и устанавливается главный индекс с именем, совпадающим с именем поля (SET ORDER TO (K)). Далее проверяется заполнение словаря и активируется POPUP-меню. При нажатии клавиши Enter в поле KODREG/KODVED заносится код В.Код видимого в словаре поля SOD из базы SLOV.DBF. Одновременно для самоконтроля пользователя в самой нижней строке экрана выводится выбранный им регион/ведомство.

Изображение экрана вместе с наложенным на него меню-словарем ведомств приведено на рис.22.14.

F2 - вызов словаря			
Код региона	Код ведомства	Название	Работников
6	1	Завод стройдеталей	351
23		Сделайте выбор	2980
		Министерство сельского хозяйства	
		Комитет по науке и образованию	

Рис.22.14

Использование кодирования длинных полей позволяет ускорить процесс обработки данных и избежать многих ошибок ввода, однако, всегда может понадобиться установить, что именно скрывается за некоторым кодом, например при выводе данных.

Процедура VIVOD реализует печать базы PREDPR.DBF, в которой коды ведомств и регионов замещаются на сами названия ведомств и регионов из файла SLOV.DBF. Для поиска нужного кода устанавливаются главный индекс (1 или 2) в области В и функцией SEEK() указатель записей переносится на нужную запись в базе SLOV.DBF. Если поиск оказался удачным (SEEK(...)=.T.), в переменные V и R пересылаются значение поля SOD из рабочей области В (т.е. из базы SLOV.DBF), если нет, выводится V/R='Не найдено'.

```
*-----Программа SLOV2.PRG-----
*-----нужны файлы PREDPR.DBF, SLOV.DBF, KODVED.IDX, KODREG.IDX-----
SET DELETED ON
CLEAR MACROS
SET ESCAPE OFF
SET TALK OFF
ON KEY
```



```

CLEAR
USE predpr IN a
USE slov IN b INDEX kodved,kodreg
DEFINE WINDOW slov FROM 5,20 TO 22,55      && Окно меню-словаря
DEFINE WINDOW predpr FROM 1,1 TO 22,61     && Окно Предприятий
DEFINE POPUP kod FROM 5,5 PROMPT FIELD b.sod;  && Меню-словарь
    TITLES 'Сделайте выбор'
ON SELECTION POPUP kod DEACTIVATE POPUP
DEFINE MENU slov      && Определение горизонтального меню
DEFINE PAD predpr OF slov PROMPT 'Предприятия' AT 0,13
DEFINE PAD vivod OF slov PROMPT 'Вывод'
DEFINE PAD slovar OF slov PROMPT 'Словари'
DEFINE PAD upak OF slov PROMPT 'Упаковка'
DEFINE PAD konec OF slov PROMPT 'Конец'
&& Определение реакций меню
ON SELECTION PAD predpr OF slov DO predpr
ON SELECTION PAD vivod OF slov DO vivod
ON PAD slovar OF slov ACTIVATE POPUP slov1
ON SELECTION PAD upak OF slov DO upak
ON SELECTION PAD konec OF slov CANCEL

DEFINE POPUP slov1 FROM 2,31      && Определение вертикального меню
DEFINE BAR 1 OF slov1 PROMPT 'Словарь ведомств'
DEFINE BAR 2 OF slov1 PROMPT 'Словарь регионов'
ON SELECTION POPUP slov1 DO slov1 WITH BAR(),PROMPT()

ACTIVATE MENU slov      && Активация горизонтального меню

PROCEDURE slov1      &&-----Процедура заполнения словарей
PARAMETERS k,p
SELECT b      && Переход в область В файла-словаря
SET ORDER TO k      && Активация нужного индекса
DEFINE WINDOW slov1 FROM 4,5 TO 23,52 && Окно заполнения словаря
GO TOP
IF EOF()      && Если такого поля в словаре нет,
    APPEND BLANK      && добавляется пустая запись и в нее
    REPLACE pole WITH ORDER()      && переносится имя поля, которое
ENDIF      && совпадает с именем индекса ORDER()
SET CARRY TO pole      && Режим копирования в новые записи поля POLE
BROWSE COLOR SCHEME 10 WINDOW slov1 TITLE p;
    FIELDS kod :H='Код',;
    sod :H='Содержание'      && Окно ввода

SET CARRY OFF
RELEASE WINDOWS slov1
RETURN

PROCEDURE predpr      &&--Процедура ввода-редактирования Предприятий
SELECT a
SET FUNCTION f2 TO '':
BROWSE WINDOW predpr COLOR SCHEME 10 TITLE 'F2 - вызов словаря';
    FIELDS kodreg :H='Код региона' :V=kod() :F,;
    kodved :H='Код ведомства' :V=kod() :F,;
    naz :H='Название',;
    rab :H='Работников'

CLEAR MACROS
RETURN

PROCEDURE upak      &&-----Процедура упаковки
SELECT a      && Упаковка основной базы данных
PACK
SELECT b      && Упаковка базы-словаря
PACK
RETURN

FUNCTION kod      &&-----Функция вызова словаря
CLEAR MACROS
IF LASTKEY()=-1      && Если нажата F2 -
    k=VARREAD()      && Запоминается имя текущего поля
    SELECT b      && Переход в область В (SLOV.DBF)
    SET ORDER TO (k)      && Активируется индекс с этим же именем
    GO TOP
    IF EOF()      && Если база пуста -
        WAIT 'Словарь пуст' WINDOW      && выдается сообщение
    ELSE      && Иначе -
        ACTIVATE POPUP kod      && активируется меню

```



```

@ 24,10 SAY b.sod    && Для проверки выводится кодируемое поле
REPLACE a.&k WITH b.kod    && Код пересылается в основную базу
ENDIF
SELECT a                && Возврат в область A
ENDIF
SET FUNCTION f2 TO ';'    && Восстановление клавиши F2
CLEAR
RETURN

PROCEDURE vivod          &&-----Процедура вывода
SELECT a
CLEAR
SCAN                    && Сканирование основной базы PREDPR.DBF
* поиск в словаре записей, соответствующих кодам базы PREDPR.DBF
  SELECT b
  SET ORDER TO 1          && Поиск региона
  v=IF(SEEK(a.kodved)=.T.,b.sod,'Не найдено')
  SET ORDER TO 2          && Поиск ведомства
  r=IF(SEEK(a.kodreg)=.T.,b.sod,'Не найдено')
  ?'Название',a.naz, 'Количество работников', a.rab
  ?'Ведомство',v
  ?'Регион',r
  SELECT a
ENDSCAN
RETURN
*-----Конец модуля SLOV2.PRG-----

```

При работе с программой сначала заполняются файлы-словари. Коды задаются пользователем произвольно (обычно в порядке возрастания), но так, чтобы они не повторялись. При этом можно реализовать механизм автоматического присвоения кодов, как это сделано в программе PRED1.PRG.

Заполнение основной базы PREDPR.DBF должно вестись уже с учетом установленных кодов с применением электронного словаря. Если пользователь помнит нужный код, обращение к словарю не обязательно.

22.4. Пример системы обработки данных

Сейчас рассмотрим пример более или менее законченной системы обработки данных по расчету ведомости выплат заработной платы работников с учетом подоходного налога.

Главная цель системы — обработка данных о выработках бригад предприятия (файлы BRIGxx.DBF) и формирование ведомостей зарплаты с учетом личных данных работников из файла KADR.DBF. Система КАДРЫ является составной частью (подсистемой) системы ВЕДОМОСТЬ и здесь только вызывается (DO KADR).

Вызов системы осуществляется командой DO VED.

Состав и функции системы

1. Базы данных системы (файлы типа DBF, FPT, IDX):

KADR.DBF и KADR.FPT — сведения о кадровом составе;

KADRTAD.IDX — индекс файла KADR.DBF по полю TAB (табель);

BRIG1.DBF, BRIG2.DBF и т.д. — сведения о месячной зарплате (выработке) работников бригады 1, бригады 2 и т.д.;

BRIG.DBF — пустой файл-эталон для создания новых бригадных файлов;

NALOG.DBF — нормативные данные о величине налогов;

NBRIG.DBF — файл номеров, обрабатываемых бригад.

2. Функции системы. Система выполняет две основные функции:

выдачу ведомостей зарплаты по бригадам;

выдачу справок о кадровом составе (подсистема КАДРЫ).

3. Программы.

VED.PRГ — основной модуль системы,
KADR.PRГ — модуль подсистемы КАДРЫ.

Формирование ведомости осуществляется после ввода данных о бригадных выработках (файлы вида — BRIG<номер бригады>.DBF) в соответствии с нормативными сведениями о подоходном налоге (файл NALOG.DBF) и льготах по налогообложению с лиц, имеющих детей/иждивенцев.

Поскольку технология налогообложения в стране стремительно меняется, меняются и цифры зарплаты и нормативные величины, читателя не должен удивлять, по всей видимости уже устаревший к моменту выхода книги, механизм расчетов и "маленькая" зарплата. Кроме того, сама эта технология в разных бухгалтериях понимается по-разному. Здесь и не ставится задача реального расчета зарплаты. При разработке системы сделан ряд допущений и упрощений относительно техники расчета. Все выплаты и налоги определяются только на текущий месяц без учета какой-либо предыстории. Не предусмотрено никакого перерасчета налогов на конец года. Естественно, такая программа может быть использована только в качестве учебной.

Рассмотрим подробнее состав и работу системы.

Модуль VED.PRГ. Это главный модуль системы. Здесь устанавливается операционная среда системы и открываются базы данных. База KADR.DBF с индексом KADRTAB.IDX в области A, база налогов NALOG.DBF — в области D, база номеров бригад NBRIG.DBF — в E. Область C резервируется под файлы бригадных выработок. Далее описываются необходимые окна. Окно NALOG — для предъявления базы NALOG.DBF, окно BRIG — для бригад, окно VEDOM — для отображения готовых ведомостей. Затем описываются главное и два вспомогательных POPUP-меню. Меню NBRIG — для отображения номеров бригад, меню VEDOM — для предъявления всех ведомостей.

Описание необходимых объектов и открытие всех файлов в самом начале программы, конечно, удобно для программиста, но требует больших ресурсов памяти. Поэтому в больших системах это делается не сразу, а по мере необходимости.

Вид главного меню со всеми вспомогательными и вызываемыми меню приведен на рис.22.15.

Кадры	Бригады	Налоги	Печать	Сервис	Расчет	Конец
Сотрудники Номера бригад	1 3 10		D: D:\VED [...] 91-6.N1 91-6.N3 91-6.N10	Упаковка данных Восстановление		

Рис.22.15

Здесь пункты горизонтального меню имеют следующий смысл:

Пункт КАДРЫ — реализует работу с данными о кадрах. В нем пункт СОТРУДНИКИ реализует работу с базой KADR.DBF, пункт НОМЕРА БРИГАД — с базой NBRIG.DBF.

Пункт БРИГАДЫ — позволяет установить номера бригадам.

Пункт НАЛОГИ — организует доступ к нормативным данным о подоходном налоге.

Пункт ПЕЧАТЬ — просмотр и печать готовых ведомостей зарплаты.
 Пункт СЕРВИС — упаковка и переиндексация баз.
 Пункт РАСЧЕТ — расчет данных и формирование ведомостей.
 Пункт КОНЕЦ — выход из системы.

Связь компонентов меню с модулями программы и данными изображена на рис.22.16.

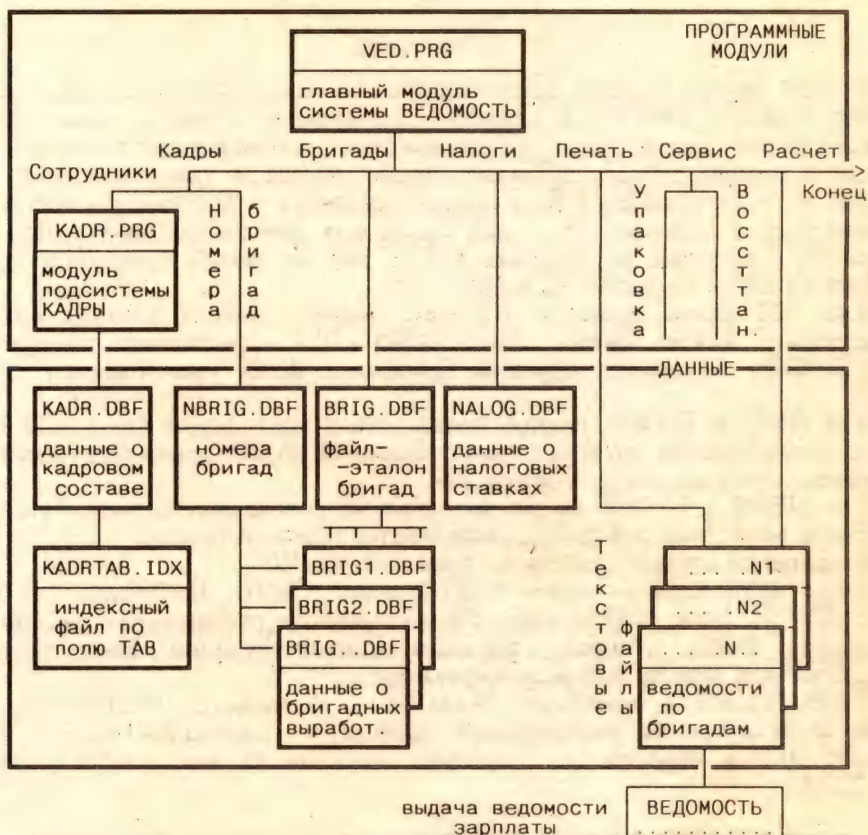


Рис.22.16

Последовательно рассмотрим все элементы меню. Если с пунктом главного меню используется пункт вспомогательного меню они соединяются знаком плюс. Пункт КАДРЫ (реализуется процедурой KDR).

Кадры+Сотрудники — осуществляется переход в область А и загружается программный модуль (DO KADR.PRГ). Такой модуль может быть построен любым способом и здесь не рассматривается.

Кадры+Номера бригад — в области Е осуществляется работа с файлом номеров бригад NBRIG.DBF. Здесь пользователь при необходимости добавляет-удаляет бригады и изменяет их номера. Эта база имеет два числовых поля длиной два разряда целых. Поле NS предназначено для хранения старых номеров бригад (бригад имеющихся на диске), а поле NN — для установления новых номеров.

Перед работой база очищается (ZAP). Затем просматриваются все файлы на диске вида BRIG<номер бригады>.DBF. Номера всех таких бригадных файлов (L) заносятся в оба поля NS и NN базы NBRIG.DBF и по команде BROWSE

предъявляются на редактирование в форме, например показанной на рис.22.17.

Номера Старые	бригад Новые
1	1
3	3
10	10

Рис.22.17

Столбец старых номеров заблокирован для редактирования и перемещения в него курсора (:W=F.) и является справочным. Столбец новых номеров первоначально имеет такое же содержимое, но пользователь может здесь изменять номера бригад, добавлять новые номера и удалять старые. После выхода из редактирования база просматривается и выполняются действия по приведению в соответствие с ней бригадных файлов на диске. Для этого создаются текстовые переменные (BS и BN) — имена бригадных файлов, сформированные из полей NS и NN.

Если NN равно нулю, а NS нет, значит, данная бригада более не существует. Такая запись базы NBRIG.DBF помечается на удаление (DELETED), а соответствующий бригадный файл уничтожается (ERASE (BS)).

Если NS=0 и NN#0, значит, пользователь ввел новый бригадный номер. Тогда копированием пустого файла-эталона BRIG.DBF создается новый файл с именем, содержащим этот номер BN.

Если NS#0 и NN#0, но их значения не совпадают, значит, бригада BS получила новое имя-номер BN. Выполняется переименование.

Завершается процесс упаковки базы NBRIG.DBF.

Пункт БРИГАДЫ — вызов POPUP-меню NBRIG. На экране появляется поле NN из базы NBRIG.DBF. Обработывается пользовательский выбор в процедуре BRIG, которая открывает соответствующий файл бригадных выработок для заполнения-редактирования.

Пункт НАЛОГИ (процедура NALOG) — вызывается BROWSE-окно для ввода-редактирования нормативных данных о подоходном налоге в базе NALOG.DBF в области D. Структура базы имеет вид, изображенный на рис.22.18.

Содержание	Название	Тип	Длина	Дес. разр.
1. Верхний предел зарплаты	ZR	Numeric	6	
2. Начальное значение налога	NZN	Numeric	9	2
3. Процент отчислений	PR	Numeric	3	2

Рис.22.18

В базе содержатся нормы вычисления налогов применительно к месячной зарплате. Эти числа пересчитаны по годовым нормам. Для России такая таблица представлена на рис.22.19.

Самое верхнее левое число — минимальное значение зарплаты (например 342 руб.). Эта сумма налогом не облагается. Кроме того, на каждого иждивенца делается скидка по налогообложению на величину минимальной зарплаты.

Конечно, к моменту выхода книги из печати, вследствие инфляции изменится зарплата и сами налоговые нормативы. Однако последнее очень легко поправить, вызвав на редактирование базу NALOG.DBF.

Предел.	Подходящий налог	
	Нач. налог	Процент
342	0.00	.00
3500	0.00	.12
7000	420.00	.15
10000	945.00	.20
15000	1545.00	.30
25000	3045.00	.40
35000	7045.00	.50
0	12045.00	.60

Рис.22.19

После того как найдена нужная строчка в базе (Зарплата \leq ZR), подоходный налог (без учета иждивенцев) вычисляется по формуле

$$NZN + (Зарплата - ZR) * PR$$

причем предел зарплаты (ZR) берется из предыдущей строки. Пусть зарплата составила 4000 руб. Тогда налог $420 + (4000 - 3500) * 0.15 = 495$

Пункт ПЕЧАТЬ (процедура VEDOM) — вызывается POPUP-меню VEDOM с перечнем текстовых файлов, содержащих ведомости зарплат для всех бригад. Выбор файла обрабатывается в процедуре VEDOM, куда передаются два параметра: BRG=PROMPT() — имя выбранного файла, KKK=LASTKEY() — код клавиши, нажатой при выборе.

Нажатие клавиши Enter (kkk=13) в POPUP-меню осуществляет вызов соответствующего текстового файла в окно VEDOM для просмотра (возможность редактирования заблокирована опцией NOEDIT).

Нажатие клавиши Space (kkk=32) вызывает опрос готовности принтера (PRINTSTATUS()) Если он готов, файл печатается (TYPE (BRG) TO PRINT), если нет — выдается соответствующее сообщение.

Пункт СЕРВИС (процедура SERV)

Сервис+Упаковка — выполняется упаковка всех файлов бригадных выработок в области С. База NALOG.DBF упаковывается в процедуре NALOG.

Сервис+Восстановление — выполняется восстановление индексного файла KADRTAB.IDX для базы KADR.DBF. Это может понадобиться после аварийного прекращения программы, например при отключении питания.

Пункт КОНЕЦ — выход из системы.

Пункт РАСЧЕТ — вызывается процедура VED для расчета выплат, удержаний и формирования файлов-ведомостей.

Рассмотрим ее работу. Здесь осуществляется расчет и формирование ведомости зарплаты для каждой из бригад. В начале на запрос программы вводится дата периода, за который выполняется расчет — год и месяц зарплаты (числовые переменные GZ и MZ).

Для вычисления удержаний необходимы данные из файла NALOG.DBF. Поскольку при расчетах эти данные нужны постоянно, лучше (быстрее) работать не с файлом, а с массивом. Для этого все записи из базы копируются в массив N по команде COPY TO ARRAY N. Такой массив будет иметь размерность Mx3, где M — число записей в NALOG.DBF. Столбцы массива будут соответствовать полям $N(i,1)=ZR$, $N(i,2)=NZN$, $N(i,3)=PR$. Элемент $N(1,1)$ содержит минимальную зарплату.

В цикле FOR на диске ведется последовательный перебор всех существующих бригадных файлов (до 20). Если очередной такой файл найден, то на экран выдается номер бригады (ii) и, поскольку результаты расчетов

выводятся в текстовый файл, экран для выдач не используется (SET CONSOL OFF). Каждому текстовому файлу дается имя по следующей схеме:

<год>-<месяц>.N<номер бригады>

Например, файл 92-05.N3 будет соответствовать ведомости бригады номер 3 за май 1992 г. Такие имена дадут возможность пользователю в дальнейшем легко распознать нужные документы. Указанное имя формируется в переменной BRG. Затем с этим именем создается и открывается текстовый (альтернативный) файл (по командам SET ALTERNATE TO (brg) и SET ALTERNATE ON). Эти команды будут рассмотрены позже.

При формировании ведомости необходимо суммирование и по вертикали, для чего вводятся переменные: SVIR — суммарная выработка всех работников бригады, SPN — подоходный налог, SNR — итоговая сумма к выдаче на руки.

В программе последовательно сканируются все записи бригадных файлов. В базе KADR.DBF разыскивается запись с тем же табельным номером (SEEK(c.tab,1)). Если такая запись не обнаружена, выдается сообщение о том, что табель не найден и анализируется следующая запись. В переменной Z вычисляется та часть зарплаты, с которой удерживается подоходный налог:

$z = \text{MAX}(0, \text{INT}(c.vir - \text{MIN}(6, a.det) * N(1,2)))$.

Налогом не облагается минимальная зарплата ($N(1,2)$), умноженная на количество детей (иждивенцев), но не более шести ($\text{MIN}(6, a.det) * N(1,2)$). Чтобы результат не получился отрицательным, если детей много, а выработка мала, используется функция MIN(). Функция INT() выделяет целую часть полученного результата, поскольку налог удерживается от суммы без копеек.

Далее вычисляется подоходный налог PN. Сначала рассматриваются два крайних случая: зарплата меньше минимальной $N(1,2)$ и больше максимальной $N(m-1,1)$. Если это не так, перебираются все строки массива N, кроме первой и последней, находится нужная строка ($z \leq n(i,1)$) и определяется PN как сумма начального налога ($N(i,2)$) плюс разница между зарплатой и суммой, от которой налог уже учтен ($Z - N(i-1,1)$), умноженная на установленный процент ($N(i,3)$):

$pn = \text{ROUND}(n(i,2) + (z - n(i-1,1)) * n(i,3), 2)$

Все вычисления выполняются (округляются) с точностью в два дробных разряда. Затем вычисляется сумма к выдаче на руки и печатается строка ведомости. По завершении расчетов для данной бригады, альтернативный файл закрывается, и процесс повторяется до обработки всех бригад.

Образец такой ведомости приведен на рис.22.20. Мы можем просмотреть и напечатать ее в пункте ПЕЧАТЬ главного меню.

Ведомость зарплаты в бригаде 1 за 1991г. месяц - 3

Фамилия	Табель	Выработка	Налог	На руки
ПОТАПОВ Д.П.	98	5000.00	491.10	4508.90
КУЛАКОВА М.И.	6	2800.00	212.88	2587.12
СИДОРОВ П.	13	100.00	0.00	100.00
МИРОНОВ Р.И	468	600.00	30.96	569.04
ВСЕГО		8500.00	734.94	7765.06

Рис.22.20

-----Программа VED.PRГ-----

SET DELETE ON
SET SAFETY OFF
SET DATE GERMAN
SET HEADING OFF


```

SET TALK OFF
CLEAR
USE kadr IN A INDEX kadrtab      && Открытие баз данных
USE nalog IN D
USE nbrig IN E
DEFINE WINDOW nalog FROM 4,15 TO 23,75      && Определение окон
DEFINE WINDOW brig FROM 4,32 TO 23,51
DEFINE WINDOW vedom FROM 1,3 TO 24,76 TITLE 'Esc - выход'
DEFINE MENU zar                      && Определение меню
DEFINE PAD kadr OF zar PROMPT '\<Кадры' AT 0,7;
MESSAGE 'Данные о кадровом составе'
DEFINE PAD brig OF zar PROMPT '\<Бригады'
DEFINE PAD nalog OF zar PROMPT '\<Налоги';
MESSAGE 'Налоговые нормативы'
DEFINE PAD PRIN OF zar PROMPT '\<Печать'
DEFINE PAD serv OF zar PROMPT '\<Сервис'
DEFINE PAD ras OF zar PROMPT '\<Расчет';
MESSAGE 'Расчет ведомостей'
DEFINE PAD konec OF zar PROMPT 'K\<онец';
MESSAGE 'Выход из системы'
ON PAD kadr OF zar ACTIVATE POPUP kadr
ON PAD brig OF zar ACTIVATE POPUP nbrig
ON SELECTION PAD nalog OF zar DO nalog
ON PAD PRIN OF zar ACTIVATE POPUP vedom
ON PAD serv OF zar ACTIVATE POPUP serv
ON SELECTION PAD ras OF zar DO ras
ON SELECTION PAD konec OF zar CANCEL
DEFINE POPUP nbrig FROM 2,15 PROMPT FIELD e.nn;
MESSAGE 'Выбор номеров бригад (Esc выход)'
ON SELECTION POPUP nbrig DO brig WITH PROMPT()
DEFINE POPUP vedom FROM 2,25 PROMPT FILES LIKE *.n* MESSAGE:
'Ведомости (Esc - выход, Enter - просмотр, Space - печать)'
ON SELECTION POPUP vedom DO vedom WITH PROMPT().LASTKEY()
DEFINE POPUP kadr FROM 2,1
DEFINE BAR 1 OF kadr PROMPT '\<Сотрудники'
DEFINE BAR 2 OF kadr PROMPT '\<Номера бригад'
ON SELECTION POPUP kadr DO kdr WITH BAR()
DEFINE POPUP serv FROM 2,38
DEFINE BAR 1 OF serv PROMPT '\<Упаковка данных'
DEFINE BAR 2 OF serv PROMPT '\<Восстановление'
ON SELECTION POPUP serv DO serv WITH BAR()
*-----Определение закончено-----
ACTIVATE MENU zar                  && Активация меню

PROCEDURE kdr                      &&-----Процедура работы с данными о кадрах
PARAMETERS i
DO CASE
CASE i=1                          && Кадры
SELECT A
DO kadr
CASE i=2                          && Номера бригад
SELECT E                          && Переход в базу номеров бригад
ZAP
FOR i=1 TO 20                    && Просмотр диска
b='brig'+LTRIM(STR(i))+'.dbf'
IF FILE(b)                      && Если бригада с таким номером есть,
APPEND BLANK                    && ее номер заносится в поля NS,NN
REPLACE ns WITH i, nn WITH i
ENDIF
ENDFOR
* Редактирование номеров бригад
BROWSE FIELD ns :H='Старые' :W=.F.:
nn :H='Новые' :B=1,20;
TITLE 'Номера бригад' WINDOW brig COLOR SCHEME 10
SCAN                            && Сканирование файла номеров бригад
bn='brig'+LTRIM(STR(e.nn))+'.dbf'
bs='brig'+LTRIM(STR(e.ns))+'.dbf'
DO CASE
CASE ns=0.AND.nn=0              && Пустая запись
DELETE                          && удаляется
CASE ns#0.AND.nn=0              && Старый номер
DELETE                          && удаляется
ERASE (bs)                      && Удаляется файл бригады
CASE ns=0.AND.nn#0              && Добавляется новый файл
COPY FILE brig.dbf TO (bn)

```



```

CASE ns#0.AND.nn#0.AND.ns#nn  && Если бригада получает
RENAME (bs) TO (bn)          && новый номер, файл
                              && переименовывается
ENDCASE
ENDSCAN
PACK                          && Упаковка
ENDCASE
RETURN

PROCEDURE nalog              &&-----Процедура ввода норм подоходного налога
SELECT D
BROWSE FIELD zr :H='Предел. зарпл.';
          nzn :H='Нач. налог';
          pr :H='Процент';
TITLE 'Подоходный налог' WINDOW nalog COLOR SCHEME 10
PACK
RETURN                      && Упаковка базы NALOG.DBF

PROCEDURE serv              &&-----Процедура СЕРВИС
PARAMETERS i
DO CASE
CASE i=1                    && Упаковка бригадных файлов
SELECT C
FOR k=1 TO 20 && Перебор бригад
b='brig'+LTRIM(STR(k))
IF FILE('b')
USE (B)
PACK
ENDIF
ENDFOR
CASE i=2                    && Восстановление (переиндексация) базы KADR.DBF
SELECT A
REINDEX
ENDCASE
RETURN

PROCEDURE brig             &&-----Процедура ввода бригадных выработок
PARAMETER nb
b='brig'+LTRIM(nb)
SELECT C
USE (b)
BROWSE FIELD tab :H='Табель';
          vir :H='Выработка';
TITLE 'Бригада '+nb WINDOW brig COLOR SCHEME 10
RETURN

PROCEDURE vedom            &&-----Процедура просмотра-печати ведомостей
PARAMETER brg, kkk
DO CASE
CASE kkk=13                && Нажатие Enter - просмотр
MODIFY FILE (brg) WINDOW vedom NOEDIT
CASE kkk=32                && Нажатие Space - печать
IF PRINTSTATUS()          && Если принтер готов,
TYPE (brg) TO PRINT        && печать
ELSE                        && Если нет - сообщение
WAIT 'Подготовьте принтер' WINDOW
ENDIF
ENDCASE
RETURN

PROCEDURE ras              &&-----Процедура расчета ведомости зарплаты
CLEAR
STORE 0 TO gz, mz          && Ввод даты начисления зарплаты
@ 10, 17 SAY 'Введите год и месяц начисления зарплаты';
          GET gz PICTURE '##' RANGE 92
@ 10, $+1 GET mz PICTURE '##' RANGE 1, 12
READ
IF LASTKEY()=27            && Если нажата клавиша Escape - выход
CLEAR
RETURN
ENDIF
SELECT D
m=RECCOUNT()              && Количество строк в базе NALOG.DBF
COPY TO ARRAY n            && Копирование базы NALOG.DBF в массив N
SELECT C

```



```

FOR ii=1 TO 20                                && Перебор бригадных-файлов
  br='brig'+LTRIM(STR(ii))+'.dbf'
  IF FILE(br)                                && Если такой файл есть,
    USE (br)                                && он открывается
  ELSE                                        && Если нет -
    LOOP                                    && переход к следующему номеру
  ENDIF
  SET CONSOLE ON
  @ 12,27 SAY 'Расчет бригады номер '+STR(ii,2) COLOR r/w
  SET CONSOLE OFF
  brg=LTRIM(STR(gz,2))+ '-' + LTRIM(STR(mz,2)) + '.N' + LTRIM(STR(ii))
  SET ALTERNATE TO (brg)
  SET ALTERNATE ON
  STORE 0 TO svir,spn,snr                    && Обнуление сумм
  ? ' Ведомость зарплаты в бригаде',STR(ii,2),,
    за 19'+STR(gz,2)+' г.    месяц -',STR(mz,2)
  ? REPLICATE('-',61)
  ? 'Фамилия                Табель Выработка    Налог    На руки'
  ? REPLICATE('-',61)
  SCAN
    IF !SEEK(c.tab,1)                        && Если в базе KADR не найден табель,
      ? 'Не найден табель',c.tab            && выдается сообщение и
      LOOP                                  && вычисления для этого работника прекращаются
    ENDIF
    z=MAX(0,INT(c.vir-MIN(6,a.det)*n(1,1))) && Зарплата, облагаемая налогом
    DO CASE
      CASE z<=n(1,1)                        && Если зарплата меньше нижнего значения,
        pn=0                                && облагаемого налогом, налог=0
      CASE z>=N(m-1,1)                      && Если больше верхнего значения
        pn=ROUND(n(m,2)+(z-n(m-1,1))*n(m,3),2)
      OTHERWISE                             && Иначе ищется нужный диапазон
        FOR i=2 TO m-1
          IF z<=n(i,1)
            pn=ROUND(n(i,2)+(z-n(i-1,1))*n(i,3),2)
          EXIT
        ENDIF
      ENDFOR
    ENDCASE
    nr=c.vir-pn                             && На руки
    ? a.fam, c.tab, ' ',c.vir,,
      TRANSFORM(pn,'#####.##'),TRANSFORM(nr,'#####.##')
    svir=svir+c.vir
    spn=spn+pn
    snr=snr+nr
  ENDSCAN
  ? REPLICATE('-',61)
  ? 'Всего',SPACE(22),TRANSFORM(svir,'#####.##'),,
    TRANSFORM(spn,'#####.##'), TRANSFORM(snr,'#####.##')
  ? REPLICATE('-',61)
  ?
  CLOSE ALTERNATE
ENDFOR
SET CONSOLE. ON
CLEAR
RETURN
*-----Конец модуля VED.PRG-----

```


Глава 23. НАДЕЖНОСТЬ СИСТЕМ ОБРАБОТКИ ДАННЫХ

Разработчик системы обработки данных должен с самого начала принимать меры для поддержания надежной совместной работы ЭВМ и пользователя, а также целостности данных. Неправильная реакция пользователя может вызвать самые тяжелые последствия для целостности данных. При этом нужно придерживаться следующих принципов:

- о все меню программы и указания пользователю должны быть четкими и ясными по существу и должны демонстрироваться на заметном месте экрана, т.е. восприниматься им быстро и без напряжения;

- о для того чтобы пользователь не забыл, какую цепь действий он уже совершил к данному моменту и не "заблудился" в системе, желательно сохранить на экране предыдущие меню/окна или их следы или вообще программировать интерфейс, "управляемый событиями" (интерфейс в стиле Windows);

- о при необходимости нужно создавать специальные программные справочные экраны о разрешенном поведении пользователя в данном месте системы с возможностью их вызова нажатием функциональной клавиши F1 и последующим возвратом в исходный экран;

- о по возможности предусматривать в программе постоянный входной контроль исходных данных (шаблонами, диапазонный и логический контроль);

- о обеспечивать отказ пользователя от продолжения исполняемых действий в любой момент;

- о потенциально опасные действия пользователя должны сопровождаться предупреждениями вида "Вы уверены?" и т.п.;

- о заложить в программе (а не выполнять после выхода в DOS) процедуры страхового копирования данных;

- о одни и те же данные по возможности должны вводиться однократно; если в системе какие-то уже имеющиеся данные, нужно задать снова, следует попытаться сделать это через меню;

- о предусмотреть выдачу больших по объему отчетов не непосредственно на принтер, а в текстовый файл, который пользователь далее может распечатать по своему усмотрению с уровня DOS или через какой-нибудь редактор. Это позволит избежать последствий, вызванных возможным сбоем принтера при печати.

Восстановление индексных файлов. В случае аварийного завершения программы (отключение питания, "зависание" компьютера) могут возникнуть некоторые проблемы с индексированными базами данных. В такой ситуации в первую очередь страдают индексы, так как СУБД для ускорения доступа старается разместить их в основной памяти. Вследствие потери этой информации файлы баз данных (DBF) перестают соответствовать файлам индексов (CDX/IDX), поскольку их актуальные значения были утрачены вместе с содержимым основной памяти. Совместная работа таких файлов более не будет возможна. При этом будет предъявлено одно из следующих сообщений об ошибке:

"Record is out of range" или "Record is not in index"

указывающее на несоответствие индекса фактическим данным ("Запись вне границ" и "Записи нет в индексе"). Коды всех ошибок приведены в технической документации и в HELP системы. В нашем случае это коды 5 и 20. Кроме того, при повреждении индекса возможна выдача сообщения (код 114)

"Index does not match database file. Recreate Index"

т.е. "Индекс не соответствует базе данных. Восстановите индекс".

Чтобы вернуть потерянное соответствие, следует переиндексировать базы данных по команде REINDEX. Такая возможность всегда должна быть доступна пользователю через специальный пункт (обычно центрального) меню.

Процесс обработки ошибок можно усовершенствовать, если воспользоваться командой **ON ERROR** и функцией

■ ERROR()

которая возвращает код ошибки.

Самый простой случай реализован ниже.

ON ERROR WAIT WINDOW IIF(ERROR()=5.OR.ERROR()=20.OR.ERROR()=114,;

Данные не корректны, сделайте восстановление индексов', '')

Здесь в случае обнаружения ошибки выполняется команда WAIT, в которой выясняется ее причина. Если код ошибки 5, 20 или 114, выдается указание на нужные действия. Системные сообщения об ошибках при этом более не выводятся.

Можно даже предусмотреть в специальной процедуре автоматический переход к нужному пункту меню или непосредственно включить в эту процедуру команду REINDEX.

Контроль последовательности обработки данных. Все процессы обработки информации можно разбить на три этапа: ввод/редактирование данных, вычисление/преобразование данных, формирование/печать выходных документов. Очевидно, что переработка информации должна осуществляться только в указанной последовательности. Так, бессмысленна печать выходного документа, если исходные данные были изменены, а нужные вычисления не сделаны.

Чтобы этого избежать, целесообразно прибегнуть к следующей технике. За каждым изменением данных, от которого зависит последовательность очередных действий, закрепляется одна переменная. Такой переменной первоначально присваивается, например, значение .T. ("Истина"). При изменении данных, которое фиксируется, например, с помощью функции UPDATE(), этой переменной дается противоположное значение .F. ("Ложь").

Если далее пользователем выполняются "правильные" действия по обработке данных, переменной снова присваивается значение Т. без каких-либо последствий.

Если нет, т.е. пользователем пропущен какой-то нужный этап обработки, следует сформировать на экране меню-предупреждение о предполагаемой ошибочности предпринятого им шага, например, в виде следующего текста, содержащего указание на причину и два возможных выбора:

Не сделано - <указание на пропущенное действие>
 ВЕРНУТЬСЯ НАЗАД ПРОДОЛЖИТЬ

Выбор пользователем позиции меню ВЕРНУТЬСЯ НАЗАД ведет к отказу от начатых действий с возвратом в предыдущее меню или даже непосредственно на пропущенное действие. Выбором позиции ПРОДОЛЖИТЬ пользователь подтверждает свое намерение и игнорирует предупреждение. Это может иметь смысл, если изменения в данных, которые внес пользователь ранее, имели "косметический" характер и, как он знает, они не повлияют на окончательный результат. При этом переменной возвращается значение .T..

В сложной системе обработки данных, которая включает несколько или даже много файлов MEM и DBF, для запоминания факта изменения в них данных лучше пользоваться даже не переменными, а массивами временных переменных. Это облегчает их анализ и предупреждение на экране предупреждений (ведь их может быть несколько сразу).

Здесь удобно пользоваться двумя одномерными векторами или одним

двумерным массивом, где первый столбец (логического типа) содержит собственно признаки изменений со значениями .T. или .F., а второй (текстового типа) — причины предупреждений.

Если допускается, что полное обновление информации в прикладной системе из-за изменения каких-то данных может не завершиться за один сеанс работы системы, целесообразно эти переменные сохранить в файле типа MEM с последующим вызовом при загрузке системы.

Страховое копирование данных. Сбой/отключение компьютера в момент исполнения операций в прикладной системе может вызвать гораздо более тяжелые последствия для данных, нежели потеря индекса, если в этот момент совершались необратимые действия. Положим, нам нужно увеличить зарплату всем работникам подразделения на величину премии. Если в момент исполнения команды REPLACE произошел сбой, то часть людей получит прибавку, а часть нет. Следствием повторного применения команды в следующем сеансе работы явится то, что некоторые сотрудники будут премированы дважды. Подобных примеров можно привести сколько угодно, в том числе и с потерей целых файлов. Так, для сортировки базы данных СУБД должна выполнить следующую цепочку действий: сортировка, удаление исходного файла, переименование отсортированного файла в исходный. Если после уничтожения исходного файла случился сбой, эти данные будут потеряны. Хотя останется отсортированный файл, но пользователь, скорее всего, не догадается его переименовать с уровня ДОС. В большинстве случаев вообще "на глаз" невозможно установить, что же пострадало в результате сбоя компьютера.

Неопытный программист часто совершенно не заботится о средствах восстановления среды на момент аварии. К сожалению, он начинает понимать их необходимость только после того, как безвозвратно потеряны результаты труда нескольких дней работы пользователя.

Точно локализовать дефект может только программное ведение подробного протокола действий прикладной системы, например в специальной базе данных, а также создание множества процедур по их устранению. Однако проще всего сделать страховое копирование всех или большинства данных (файлов DBF, FPT, IDX, CDX, MEM) в самом начале работы программы в специальную директорию (например, с именем STRAH), а в самой программе предусмотреть возможность их возврата в основную директорию. Страховое копирование не должно быть автоматическим, иначе при послеаварийном запуске машины программа сразу перегонит испорченные файлы в страховую директорию и восстановить данные будет не из чего.

При наличии такого средства со сбоем компьютера будут потеряны результаты только одного сеанса работы (максимум — день труда). При желании еще уменьшить потери можно периодическим копированием внутри самой программы.

Удобным представляется поддержка в программе специальной переменной (например, с именем NORMZ), фиксирующей нормальное/ненормальное завершение программы, а также переменной (NORMD) — даты текущего сеанса и сохранение их в специальном MEM-файле (с именем NORM.MEM). Переменная NORMZ должна иметь значение "Истина", если последний сеанс работы завершился нормально, и "Ложь" в противном случае. Тогда первым шагом прикладной системы должны быть считывание этого файла с диска и анализ переменной. Ниже приведены фрагменты программы, относящиеся к страховому копированию.

*-----Программа страхового копирования-----
 SET CLEAR OFF SET SAFETY OFF
 IF FILE("norm.mem")

&& Если файл NORM.MEM есть,


```

RESTORE FROM ("norm.mem")      && он загружается в память
IF normz                        && Если переменная завершения истинна,
! COPY *.dbf .\strah >NUL      && данные копируются в
! COPY *.idx .\strah >NUL      && страховую директорию
ELSE                             && Иначе - сообщение
WAIT normd+ ' было аварийное завершение.' +
      Восстановить (Enter) данные?' WINDOW
IF LASTKEY()=13                && и если нажата Enter,
! COPY .\strah\*.dbf . >NUL    && файлы восстанавливаются
! COPY .\strah\*.idx . >NUL    && из страховой директории
ENDIF
ENDIF
normz=.f.                      && Устанавливается переменная завершения
normd=DTOC(DATE())             && и переменная текущей даты
SAVE ALL LIKE norm? TO norm    && И запоминается в файле
* <----- Собственно программа----->
normz=.t.                      && Устанавливается переменная завершения
SAVE ALL LIKE norm? TO norm    && И запоминается в файле
CLEAR
IF MOD(DAY((DATE())) ,3)=0      && Если дата кратна трем,
  @ 10,9 SAY:
  'Сегодня необходимо выполнить копирование данных на дискеты';
  COLOR W*/R                    && выдается сообщение
ENDIF
*-----Конец программы-----

```

Здесь сначала выясняется, есть ли файл NORM.MEM в текущей директории. Если есть, файл загружается и выясняется значение переменной NORM. При NORMZ=.T. все файлы данных копируются в страховую поддиректорию внутри рабочей директории (.\STRAH). Поскольку файлов в системе обычно довольно много, проще использовать не собственные команды копирования FoxPro, а выполнить (через знак "!") соответствующие команды DOS, для которых можно указать маску. Чтобы системные сообщения команд не выводились на экран, они переадресованы на "пустое" устройство ДОС — NUL. Если NORMZ=.F., значит, предыдущий сеанс работы завершился аварийно. На экран выдаются сообщение и запрос о необходимости восстановления данных из страховой директории (нажатием клавиши Enter). Такой запрос нужен для того, чтобы отказаться от копирования данных, если завершение программы было ненормальным, но данные, как знает пользователь, не пострадали. После завершения анализа переменной NORMZ она получает значение .F. и запоминается в файле NORM.MEM вместе с переменной NORMD. Далее идет тело собственно прикладной системы. Если при этом произойдет сбой, переменная NORMZ сохранит свое значение. При нормальном завершении программы этой переменной присваивается значение .T., и она сохраняется в файле.

Кроме текущего страхового копирования следует позаботиться о периодическом копировании данных на внешние носители на случай возможной "гибели" винчестера. Такую операцию, конечно, выполняют не ежедневно, а раз в несколько дней, например раз в три дня. С этой целью в самом конце программы производится анализ текущей даты. Если она кратна трем (IF MOD(DAY((DATE())) ,3)=0), пользователю выдается сообщение о желательности копирования на дискеты. С тем чтобы сообщение оставалось на экране и после окончания программы и выхода из FoxPro, в ее начале сделана соответствующая установка командой

■ SET CLEAR OFF

По умолчанию SET CLEAR ON.

Округления при вычислениях. Иногда при анализе данных, полученных на ЭВМ, возникает видимость, что вычисления сделаны неверно. Этот эффект получается вследствие гораздо большей точности компьютера по сравнению с той, с которой обычно оперирует человек. Разберем такой характерный

случай. Пусть ЭВМ нужно сложить следующий фактический ряд чисел:

$$2.69 + 3.78 + 5.18 + 7.09 = 18.74$$

Если предположить, что слагаемые и результат при выводе округляются до одного дробного разряда, то мы увидим на экране

$$2.7 + 3.8 + 5.2 + 7.1 = 18.7$$

Хотя с учетом округления этот результат правильный, ручные вычисления его не подтверждают (сумма равна 18.8), поскольку человеку неизвестны значения второго округленного разряда и он, естественно, считает их равными нулю.

При полностью ручном счете такая ситуация невозможна, так как в течение всего процесса вычислений производится округление всех операндов еще до их суммирования и результат будет не 18.7, а 18.8.

В этом случае и в программе необходимо выполнять округления не при выдаче готовых данных, а постоянно. Последнее характерно, например, в денежных документах. Так, если в рассмотренном выше примере имеются в виду выплаты работникам предприятия, то, если не принять мер по своевременному округлению данных, получится, что кассир должен получить к выплате 18.7 руб., а выдать — 18.8 руб., что, конечно, недопустимо.

Создание контекстно-зависимой экранной подсказки HELP. Для каждого случая, когда пользователю, вероятно, понадобится помощь, можно писать свои процедуры помощи, как это сделано в программе KADR1.PRG (процедура F1). Однако в больших системах таких экранов может оказаться довольно много, что потребует непродуктивных затрат памяти для процедур помощи. В этом случае лучше хранить экраны подсказки в базе данных.

Очень простой механизм помощи реализуется следующим образом. Сами тексты помощи предварительно заносятся в файл, который мы назовем HELP.DBF. Эта база состоит из единственного мемо-поля HLP. В прикладной системе он открывается, например, в области G. Кроме того, для экрана помощи определяется окно также с именем HELP, а клавиша F1 закрепляется за процедурой с именем HELP, в которую передается один параметр K.

```
SET ESCAPE OFF
ON KEY LABEL f1 DO help WITH k
DEFINE WINDOW help FROM 1,1 TO 10,25
USE help IN g
```

Этот параметр в дальнейшем будет указывать номер записи в файле HELP.DBF, где содержится нужная подсказка. В самой процедуре (ниже) осуществляется перевод указателя записей в области G на запись номер K.

```
PROCEDURE help
PARAMETERS k
IF k#0
GO k IN g
MODIFY MEMO g.hlp WINDOW help NOEDIT
ENDIF
RETURN
```

Далее по команде MODIFY MEMO открывается окно HELP, в которое и выводится текст подсказки. Чтобы исключить случайную возможность порчи содержимого HELP пользователем, возможность редактирования исключена опцией NOEDIT. Такой HELP, если он обширный, можно листать в открытом для него окне.

Теперь из любого места программы можно вызвать свою помощь нажатием клавиши F1. Доступ к нужной записи в файле помощи происходит очень быстро, поскольку ее не приходится искать. Важно только не забывать в соответствующих местах программы делать присвоения переменной K, т.е.

указывать номер нужной записи из файла HELP.DBF. В процедурах, где помощь не предусмотрена, переменной K следует задавать значение 0.

Если помощь требуется "привязать" не к процедурам, а к конкретным редактируемым полям, то в файл HELP.DBF нужно ввести дополнительное поле, содержащее имена полей, откуда может прийти вызов, а само обращение за помощью организовать по команде вида

```
ON KEY LABEL f1 DO <процедура> WITH VARREAD()
```

В процедуре HELP тогда сначала организуется поиск (SEEK/LOCATE) записи, содержащей имя нужного поля, где находится экран помощи.

Можно обойтись и без дополнительного поля, если в процедуре помощи в структуре DO CASE организовать доступ к записи, закрепленной за полем. Например, если помощь для поля FAM содержится в записи номер 4, то в процедуре должны быть строки CASE x='FAM' и GO 4, где переменная X — формальный параметр, получающий значение VARREAD().

Экраны подсказки для полей можно вызывать и с помощью опций W и WHEN команд BROWSE и @...GET. Эти опции контролируют входы в соответствующие поля/переменные. В них легко осуществить вызов ПФ, содержащих активацию окон, и выдачу в них нужных сообщений-подсказок или задание нужного номера записи K.

Использование текстовых файлов для сохранения отчетов. В СУБД имеются средства запоминания результатов работы команд выдачи данных в указанном программистом текстовом файле. Применение текстового файла позволяет разделить процессы генерации и печати выходных документов.

Зачем это нужно? Почему не всегда сразу удобно направлять готовые данные на принтер? Есть несколько причин.

1. Если вырабатываемые данные имеют значительный объем, требуется надежная и безостановочная работа принтера в течение значительного времени. Сбой принтера или неправильная подача бумаги хотя бы даже в самом конце выдачи вынуждают пользователя запустить заново всю программу печати. Это влечет значительную потерю времени, в особенности если (как это обычно бывает) в программе печати делаются еще какие-то вычисления, трансформации данных, поиск и перемещения в базе данных. Печать же текстового файла, при которой не выполняется никаких иных действий, осуществляется значительно быстрее. Кроме того, можно печатать его частями. Легко решается проблема выдачи нескольких копий документа, в том числе и в разное время. Результаты работы системы вообще могут передаваться заинтересованным лицам непосредственно на дисках.

2. При создании прикладных систем программист очень часто не имеет сведений о том, каким принтером будет пользоваться заказчик (т.е. неизвестен не только его тип, но и ширина каретки). В особенности это характерно для систем широкого применения, которые ориентированы не на конкретного пользователя, а на открытый рынок программных продуктов. Ввиду этого целесообразно предусмотреть сохранение всех выданных, в особенности с длиной строки более минимально гарантированных любым принтером 80 символов в текстовом файле, оставив за пользователем непосредственную организацию печати на бумаге.

3. При отладке программ непосредственный просмотр выданных с длиной строки более 80 знаков на экране делается затруднительным. Альтернативой здесь является использование текстового файла и встроенного редактора FoxPro (команды MODIFY FILE/COMMAND). Техника такого процесса рассмотрена ниже.

Создание текстового (альтернативного) файла начинается с команды объявления его имени

■ SET ALTERNATE TO <имя файла> [ADDITIVE]

Если файл с таким именем уже есть, он будет уничтожен, если в команде опущена опция ADDITIVE. По умолчанию новый файл получает расширение TXT. Команда SET ALTRNATE TO без параметра указывает, что работа с файлом закончена, и делает его более недоступным для дополнения. Аналогичное действие оказывает команда

■ CLOSE ALTERNATE

После объявления имени файла по другой команде

■ SET ALTERNATE ON/OFF

файл открывается (ON) для приема в него данных и закрывается, если некоторые из них нужно проигнорировать (OFF). Переключение ON/OFF может быть сделано многократно, и каждый раз новые данные, вырабатываемые командами ?/??/LIST, будут дописаны в конец предыдущих. Одновременно они выводятся и на экран. Обычно этого не требуется. Для экономии времени и сохранения экрана на данный период лучше отключать (OFF) выдачу их на экран по команде

■ SET CONSOLE OFF/ON,

не забывая ее своевременно возобновлять (ON).

Ниже приведен фрагмент программы, в которой открывается и периодически дополняется текстовый файл TTT.TXT:

SET ALTERNATE TO ttt	- объявление имени файла TTT.TXT
SET ALTERNATE ON	- открытие текстового файла
SET CONSOLE OFF	- отключение дисплея
?< занесение данных в файл >	
SET ALTERNATE OFF	- закрытие файла
SET CONSOLE ON	- активация дисплея
?< выдача только на экран >	
SET ALTERNATE ON	- открытие файла
?< выдача в файл и на экран >	
SET ALTERNATE TO	- закрытие и отключение файла
MODIFY FILE ttt.txt	- просмотр файла TTT.TXT

В программе, как правило, имена текстовым файлам отчетов даются не произвольно, а так, чтобы пользователь мог легко распознать смысл документа. Поскольку имена не могут состоять из русских слов, удобно, если это имеет смысл, давать цифровые имена, содержащие, например, дату создания файла-отчета, номер подразделения, табельный номер и т.д.

Печать текстовых файлов, выработанных программой, можно осуществлять по команде

■ TYPE <файл> TO PRINT

(при этом следует только "подавить" ненужный прогон листа и выдачу технических сообщений по команде SET HEADING OFF) или любым другим способом с уровня операционной системы.

Здесь рассмотрен самый простой механизм формирования отчета. FoxPro имеет исключительно развитые средства вывода данных, включая генератор отчетов (см. соответствующий раздел), набор драйверов принтеров и средств их формирования, а также множество специальных команд и системных переменных для обслуживания отчетов. Большинство из этих средства, кроме генератора, оставлены за рамками книги.

Потребность в дисковом пространстве. При проектировании систем обработки данных необходимо позаботиться о наличии достаточного места на винчестере. При этом нужно предусмотреть место как для данных и программы, так и для временных файлов FoxPro.

Временные файлы создаются системой только на время ее работы. Такие файлы получают случайные имена из произвольных алфавитно-цифровых символов и имеют (для сетевой версии) расширение имени TMP. После нормального завершения сеанса они автоматически уничтожаются. При аварийном завершении файлы остаются на диске и должны быть уничтожены самим пользователем с уровня DOS. О содержании этих технических файлов известно довольно мало, но размеры их могут быть большими. Например, при сортировке может понадобиться дисковое пространство втрое больше размера исходного файла.

Файлы данных — это прежде всего DBF-, IDX-, CDX- и FPT-файлы. Размер собственно базы данных (DBF-файла) легко вычислить, умножив сумму длин всех полей записи на предполагаемое число записей. Остается неучтенным заголовок файла, но он невелик и может быть проигнорирован. Размер обычного индексного IDX-файла можно определить, умножив число проиндексированных записей на длину индексного ключа плюс примерно 30% для указания в нем номеров записей из основной базы и технической информации. Размер компактного IDX-файла зависит от вида данных и может меняться. В худшем случае его можно считать равным числу записей, умноженному на длину ключа. Индексный CDX-файл практически является суммой соответствующих его тегам компактных индексных файлов. Размер FPT-файла (файла для мемо-полей) прогнозировать очень сложно. Его длина определяется фактическими вводами данных в мемо-поля. При этом дисковое пространство отводится порциями, по умолчанию — 64 байта.

Учитывая вышесказанное, можно сделать лишь грубую прикидку требуемой дисковой памяти.

При работе уже готовой системы, с помощью специальных функций FoxPro (ADIR(), DISKSPACE() и др.) программным образом можно контролировать как размер свободной памяти, так и размеры файлов, не допуская критических ситуаций. Если все-таки дискового пространства не хватило, FoxPro выдаст сообщение

"Notenough disk space"

Такую ошибку можно перехватить по команде ON ERROR и распознать функцией ERROR() по ее коду — 56.

Поддержание уникальности записей. В некоторых случаях критически важно обеспечить неповторяемость данных, т.е. исключить возможность ввода записей с значением поля, которое уже есть в базе данных. Такими базами являются, например, словари. Для базы KADR.DBF обязательной является уникальность табельного номера (поля TAB).

В FoxPro имеется инструмент поддержания уникальности — опция UNIQUE в команде индексирования INDEX ON <поле> TO <индексный файл> UNIQUE. В этом случае новая запись с повторяющимся значением <поля> не будет включена в <индекс> и, таким образом, не будет доступна для просмотра и обработки. Применительно к нашей базе такая команда может выглядеть следующим образом (создается индекс TABUN.IDX):

```
INDEX ON tab TO tabun UNIQUE
```

Однако здесь есть и проблемы. Во-первых, не блокируется сам ввод повторяющихся данных; во-вторых, не генерируется никаких сообщений и, в-третьих, такая запись, хотя и не включается в индекс, в самой базе сохраняется и остается доступной при отключении уникального индекса или назначении других главных индексов. Кроме того, усложняется удаление этих записей. Таким образом, лучше программно организовывать контроль повторяющегося ввода.

Самым простым решением было бы включение в опцию проверки ввода (V/VALID) команд BROWSE/@...GET функции !SEEK(<поле>), которая бы возвращала значение .F. в случае нахождения повтора. Так именно следует и поступать, когда редактирование данных выполняется не непосредственно в базе, а сначала в некоторых "транзитных" переменных. Однако, если речь идет о команде BROWSE, этот механизм использовать нельзя, поскольку функция SEEK() начинает поиск именно с текущей записи, так что он всегда будет успешным. Поэтому для организации контроля придется использовать более сложную технику поиска, игнорирующую текущую запись. Далее приведен фрагмент программы, реализующий эту задачу.

```
USE kadr INDEX kadrtab
BROWSE FIELDS fam,tab :V=tab=0.OR.un() :E='Такая запись в базе уже есть'

FUNCTION un                &&-----Функция проверки уникальности
n=RECNO()                 && Запоминается номер редактируемой записи
t=tab                     && и значение табельного номера
SEEK tab                  && В базе ищется совпадающее значение
IF RECNO()=n              && Если это та же самая запись,
    SKIP                  && переход на следующую
ENDIF
IF t=tab.AND.!EOF()       && Если теперь совпадение и не конец файла,
    @ 21,6 SAY fam+STR(tab,4) && выводятся данные этой записи
    GO n                  && возврат на исходную запись,
    RETURN .f.           && выход из поля запрещается
ENDIF
@ 21,1                    && Очистка 21-й строки
GO n                      && Возврат на исходную запись
RETURN
```

Здесь используется индексный файл KADRTAB.IDX, созданный без параметра UNIQUE. В опции V для проверки ввода в поле TAB вызывается функция UN(), если введено любое значение, кроме нуля. Возможность повторения нуля сохранена ввиду того, что в базе могут находиться несколько записей с временно пустыми полями TAB.

В процедуре-функции UN сначала запоминаются номер текущей записи и значение поля TAB, которое затем ищется в базе данных. Если в результате поиска найдена именно текущая, а не какая-то другая запись, то указатель сдвигается на следующую запись. Далее проверяется, совпадает ли поле TAB этой записи с введенным значением и не конец ли это файла. Если нет, значит, найдено поле-дубликат и данные из него выводятся для сведения пользователя в 21-й строке экрана. В опцию V по команде RETURN возвращается запрещающее значение .F. и выдается сообщение "Такая запись уже есть". В противном случае процедура завершается без последствий.

Если база не индексируется или применяется технология Rushmore, команда LOCATE выполняет поиск, игнорируя текущую запись. Фрагмент процедуры приведен ниже.

```
LOCATE FOR tab=t
IF FOUND()
    @ 21,6 SAY fam+STR(tab,4)
    GO n
    RETURN .f.
ENDIF
```


Глава 24. МАНИПУЛИРОВАНИЕ ФАЙЛАМИ

СУБД располагает развитыми средствами по программному формированию, копированию и удалению данных. Сначала рассмотрим несколько общих команд.

Просмотр имен файлов на диске выполняется по команде

■ **DIR** [<диск>] [<маска>] [TO PRINTER/TO FILE <файл>]

Команда выводит на экран/принтер/файл имена файлов из указанного диска и директории. Ее основное отличие от аналогичной команды DIR операционной системы заключается в том, что по умолчанию команда предьявляет имена не всех файлов, а только файлов DBF. Если нужно увидеть все содержимое директории, необходимо задать команду DIR *.*.

Следующая команда удаляет неактивный в данный момент файл любого типа.

■ **ERASE** <файл> или **DELETE FILE** <файл>

Имя файла должно быть указано с расширением. Одной командой может быть удален только один файл, поскольку применение маски не предусматривается, например ERASE KADR.DBF.

■ **RENAME** <старое имя файла> TO <новое имя файла>

Команда осуществляет переименование любого неактивного файла. Расширение имени обязательно.

Следующая команда выполняет копирование любого неактивного файла в новый файл с тем же или другим именем. Расширение для обоих файлов указывать обязательно.

■ **COPY FILE** <имя файла-оригинала> TO <имя файла-копии>

Например,

.COPY FILE C:\FOX\KADR.PRГ TO A:KADR.PRГ

24.1. Копирование файлов БД

Копирование в новый файл БД. Следующая команда осуществляет копирование открытого файла DBF в новый файл, который этой командой создается:

■ **COPY TO** <имя нового файла> [<границы>]

[FIELDS <поля>] [FOR <условие>] [WHILE <условие>]

[TYPE <тип файла>] [WITH CDX]

В новый файл могут копироваться как все поля базы данных, так и только перечисленные в списке FIELDS (если есть). Причем копируемые поля могут находиться не только в файле из активной рабочей области, но и в любом другом файле базы данных из других рабочих областей. В этом случае имена полей — составные (имя базы и имя поля).

Если указаны условия и/или границы, копироваться будут только удовлетворяющие им записи. По умолчанию область действия команды — весь файл (ALL).

Копирование возможно вместе со структурным индексным файлом (WITH CDX).

При отсутствии фразы TYPE новый файл будет также файлом базы данных с расширением имени DBF.

Указание типа позволит сформировать новые файлы иных типов, пригодные для просмотра и редактирования в текстовых редакторах и обработки в других средах. При этом каждая запись файла-оригинала в новом

текстовом файле будет завершаться символом возврата каретки и переводом строки, даты будут выведены в форме ГГГГММДД, и все поля будут разграничены в соответствии с параметром, установленным после слова TYPE.

Допустимы следующие значения этого параметра.

SDF — устанавливается системный формат данных (System Data Format), где все поля остаются заданной в базе данных длины без разграничителей. Все строки — также одинаковой длины.

Кроме указанной стандартной формы ASCII-файла возможно задание типа файла-копии, пригодного для обработки в пакетах VisiCalc, Multiplan, Lotus 1-2-3, Excel (см. документацию). Можно также установить собственные разделители данных с помощью фразы

DELIMITED [WITH <разделитель>/BLANK/TAB]

а именно:

DELIMITED — поля разделяются запятыми, символьные поля берутся в двойные кавычки;

DELIMITED WITH BLANK — поля отделяются пробелами;

DELIMITED WITH TAB — поля отделяются знаками табуляции;

DELIMITED WITH <разделитель> — все поля отделяются запятыми, символьные поля берутся в указанные <разделители>.

По умолчанию новый файл будет иметь расширение TXT.

Пример. Из файла KADR.DBF создать новый файл, включающий поля FAM, SEM, DET для всех работников, родившихся в 1946 г. и позже. Назовем этот файл именем KADR1946.DBF.

```
.SET DATE GERMAN
.USE kadr
.COPY TO kadr1946 FIELDS fam,sem,det FOR dtr>={01.01.46}
.USE kadr1946
.LIST OFF
```

Пример. Скопируем третью запись из файла KADR.DBF в текстовые файлы с разными разделителями.

```
USE KADR
COPY TO a RECORD 3 FIEL fam,dtr,tab,pol TYPE SDF
COPY TO b RECORD 3 FIEL fam,dtr,tab,pol TYPE DELIMITED
COPY TO c RECORD 3 FIEL fam,dtr,tab,pol TYPE DELIMITED WITH BLANK
COPY TO d RECORD 3 FIEL fam,dtr,tab,pol TYPE DELIMITED WITH *
```

Последовательный просмотр полученных текстовых файлов A.TXT, B.TXT, C.TXT, D.TXT дал следующие результаты:

```
A - КУЛАКОВА М.И. 19490415 6Ж
B - "КУЛАКОВА М.И.", 19490415,6, "Ж"
C - КУЛАКОВА М.И. 19490415 6 Ж
D - *КУЛАКОВА М.И.*, 19490415,6, *Ж*
```

Данная команда может быть использована для пересылки данных из файлов DBF в текстовые файлы и для передачи их в другие системы обработки информации (СУБД, электронные таблицы, алгоритмические языки).

Копирование в существующий файл БД. Следующая команда добавляет в активную базу данные из некоторой другой базы или текстового файла:

■ APPEND FROM <имя файла-источника> [FIELDS <поля>]
[FOR <условие>] [WHILE <условие>] [TYPE <тип файла>]

Если параметр TYPE опущен, копируется файл DBF. Тогда команда является близкой по смыслу к предыдущей, но она не создает новый файл, а лишь дописывает записи в конец существующего. Из файла-источника добавляются только поля, одноименные с полями дополняемого файла. Параметр FIELDS позволяет копировать не все такие поля, а лишь перечисленные, т.е. файл-источник может иметь иную структуру.

Команда применяется и для копирования в базу данных строк файлов

любого из типов, перечисленных в команде COPY TO, а также из систем Framework, Paradox, RapidFile. При этом надо проследить, чтобы дополняемые элементы имели длины и содержимое, соответствующие длинам и типам полей базы данных. Новые записи могут быть образованы и из строк текстовых файлов.

Команда в формате APPEND FROM ... TYPE является удобным средством перенесения данных из ASCII-файла в DBF-файл. Такие данные вообще могут готовиться и вне СУБД, в любом текстовом ASCII-редакторе. Важно только, чтобы машинистка строго придерживалась установленных колонок.

При приеме файла в символьные поля данные принимаются как символьные, в числовые — как числовые. Строки, изображающие даты, в поля типа дата базы-приемника не заносятся (по крайней мере, автору это не удалось). В таком случае следует прибегнуть к копированию подобных данных в специально предусмотренное символьное поле с последующим занесением его в поле типа дата командой REPLACE (используя функцию DTOC()).

24.2. Работа со структурами БД

Копирование структуры базы данных. Во многих случаях желательно иметь возможность командным образом создавать новый файл базы данных со структурой, аналогичной или близкой некоторому другому DBF-файлу. Эта возможность предоставляется командой

■ COPY STRUCTURE TO <имя нового файла> [FIELDS <поля>]

Команда копирует структуру активной базы (или ее часть) во вновь создаваемый командой пустой файл.

Применение этой команды удобно, например, в следующих случаях. Ранее описывалась работа с файлами бригадных выработок: BRIG1.DBF, BRIG2.DBF и т.д. Часто на предприятиях бригады изменяются, возникают и расформировываются при производственной необходимости. Отсюда требуется как уничтожение, так и создание новых бригадных файлов. Поэтому целесообразно сначала создать некоторый пустой эталонный файл (назовем его BRIG.DBF) и на основе его по мере необходимости уже создавать рабочие файлы для бригад с заданными номерами.

Ниже приводится фрагмент такой программы.

```
IF !FILE(s)
  USE brig
  COPY STRUCTURE TO (s)
ENDIF
USE (s)
CHANGE
```

Пусть пользователем уже задан номер бригады, а затем программой сформировано в переменной S полное символьное имя бригады, с которой необходимо работать (эти команды не показаны). Тогда в приведенном фрагменте сначала выясняется (IF !FILE (S)), есть ли такой файл на диске. Если нет, открывается (USE BRIG) файл-эталон BRIG.DBF, структура которого копируется во вновь создаваемый файл, после чего он открывается (USE (S)) и редактируется.

Программное создание структуры БД. До сих пор структура файла базы данных создавалась только в интерактивном режиме по команде CREATE. При этом программист явно задавал для файла имена всех полей, их типы и размеры. В процессе обработки данных часто возникает необходимость в создании новых файлов баз данных, о структуре которых первоначально известно лишь в самых общих чертах. Доверять такую работу пользователю нельзя, поскольку это вовлечет его в решение следующих технических

вопросов: назначение имен полей, определение их типов и т.д. Выход может быть найден с помощью двух следующих команд.

В СУБД имеется команда, позволяющая занести структуру активного файла базы данных в некоторый другой новый файл, но уже в качестве его данных. Назовем этот файл файлом-макетом.

■ COPY TO <имя файла-макета> STRUCTURE EXTENDED

Структура такого файла-макета не будет копией структуры исходного файла, как это было, например, при использовании команды COPY STRUCTURE. Она будет иметь всегда один и тот же вид и состоять из четырех полей:

- FIELD_NAME — имя поля (символьного типа);
- FIELD_TYPE — тип поля (символьного типа);
- FIELD_LEN — длина поля (числового типа);
- FIELD_DEC — количество десятичных разрядов (числового типа).

Поля файла-макета будут содержать в качестве данных соответствующие параметры файла-источника.

Значение поля FIELD_TYPE будет указывать тип каждого из полей базы данных в виде одного из символов C, N, L, M или D (символьный, числовой, логический, мемо или тип дата соответственно).

Файл-макет — это обычный файл базы данных, содержимое которого может быть в дальнейшем изменено или дополнено программным путем, после чего на его основе может быть создан новый пустой файл базы данных по следующей команде:

■ CREATE <имя нового файла> FROM <имя файла-макета>

Файл-макет должен быть активным. После выполнения команды активным становится новый файл, а файл-макет закрывается.

Пр и м е р. Пусть требуется из файла KADR.DBF создать новый пустой файл KADRNOV.DBF с похожей структурой, но без полей POL, SEM, DET, SZAR, PODR, PER и с новым полем PREM (премия) числового типа длиной 6.2. Для того чтобы удобно было следить за процессом, здесь целесообразно активировать статус-строку.

```
.SET STATUS ON
.USE kadr
.COPY TO kdrmaket STRUCTURE EXTENDED
.USE kdrmaket
.LIST
Record # FIELD_NAME FIELD_TYPE FIELD_LEN FIELD_DEC
1 FAM C 025 000
2 DTR D 008 000
3 TAB N 003 000
4 POL C 001 000
5 SEM C 001 000
6 DET N 001 000
7 SZAR N 007 002
8 PODR C 010 000
9 PER M 010 000

.GO 4
.REPLACE field_name WITH 'prem', field_type WITH 'n';
      field_len WITH 6, field_dec WITH 2

.GO 5
.DELETE REST
.PACK
.CREATE kadrnov FROM kdrmaket
.LIST STRUCTURE
Field Field Name Type Width Dec
1 FAM Character 25
2 DTR Date 8
3 TAB Numeric 3 0
4 PREM Numeric 6 2
```

Здесь сначала создается из файла KADR.DBF файл-макет KDRMAKET.DBF, а затем после его

изменения (уничтожения пяти последних записей, содержащих реквизиты полей POL, SEM и т.д) и дополнения одной новой записью с характеристиками нового поля PREM формируется файл KADRN OV.DBF:

```
KADR.DBF -> KDRMAKET.DBF -> KADRN OV.DBF
```

Для иллюстрации процессов преобразования включены команды LIST и LIST STRUCTURE, показывающие динамику изменения данных.

Содержательная сторона рассмотренного примера довольно примитивна. Тот же результат, и даже быстрее, можно получить с помощью команды MODIFY STRUCTURE. Смысл использования файла-макета заключается в возможности его обработки обычными средствами, в том числе и в программно-управляемом диалоге с пользователем, а не директивно, как в примере. Это предоставляет средства разработки программ, дающих удобную возможность пользователю создавать новые файлы, не прибегая к определению структуры файла вручную командой CREATE <имя файла>.

С этой целью рассмотрим другую, более сложную задачу, имеющую отношение к уже знакомым базам данных, содержащим сведения о месячной выработке рабочих в бригадах (файлы BRIG...DBF).

Пример. Предположим, что в течение месяца рабочие могут работать в составе разных бригад и количество бригад из месяца в месяц может изменяться. Пусть, например, в текущем месяце было три бригады (номер 1,4 и 12) и их файлы выглядят, как показано на рис.24.1.

BRIG1.DBF			BRIG4.DBF			BRIG12.DBF		
TAB		VIR	TAB		VIR	TAB		VIR
1	98	330.00	1	13	100.50	1	10	410.60
2	6	400.00	2	50	84.00	2	98	70.40
3	13	12.00	3	6	50.00	3	468	200.00
4	468	170.00				4	50	320.60
						5	13	230.00

Рис.24.1

Целью задачи является создание из всех бригадных файлов, содержащих табельный номер (TAB) и выработку работника (VIR), нового итогового файла ITOGBRIG.DBF, который бы содержал информацию, изображенную на рис.24.2.

ИТОГ (ITOGBRIG.DBF)				
ТАБЕЛЬ	Бригады			ИТОГО
	1	4	12	
98	330.00	0.00	70.40	400.40
6	400.00	50.00	0.00	450.00
13	12.00	100.50	230.00	342.50
468	170.00	0.00	200.00	370.00
50	0.00	84.00	320.60	404.60
10	0.00	0.00	410.60	410.60
ВСЕГО:	912.00	234.50	1231.60	2378.10

Рис.24.2

Здесь любой табельный номер встречается только один раз, но в столбце каждой бригады, где работал данный работник, стоит его выработка. Кроме того, в столбце ИТОГО эти выработки суммируются. Суммирование производится также по вертикали.

Определим структуру базы ITOGBRIG.DBF. Она должна иметь поле табельного номера (TAB), поле итоговых сумм по горизонтали (ИТОГ) и поля бригадных выработок. Все поля имеют числовой тип. Поле TAB должно, как и в бригадных файлах, иметь длину 3. Поле ИТОГ — длину 7 при двух разрядах дробной части (до 9999.99 рублей). Поля, в которых будут содержаться выработки по бригадам, имеют тот же тип и размер, что и поле ИТОГ. Таким полям удобно дать имена, совпадающие с именами соответствующих бригадных файлов. Вообще говоря, имена могут

быть любыми, важно только, чтобы они содержали номера бригад (1, 4, 12). Структура базы ITOGBRIG.DBF показана ниже.

Field	Field Name	Type	Width	Dec
1	TAB	Numeric	3	0
2	ITOG	Numeric	7	2
3	BRIG1	Numeric	7	2
4	BRIG4	Numeric	7	2
5	BRIG12	Numeric	7	2

Сложность заключается в том, что количество бригад и их номера меняются из месяца в месяц. Ввиду того что мы не можем заранее создать файл с нужной структурой, необходимо написать программу, строящую такой файл в зависимости от наличия бригад.

Первое, что должен сделать программист — это создать файл-макет со следующей структурой:

Field	Field Name	Type	Width	Dec
1	FIELD_NAME	Character	10	
2	FIELD_TYPE	Character	1	
3	FIELD_LEN	Numeric	3	
4	FIELD_DEC	Numeric	3	

Назовем файл MAKETITG.DBF.

При создании структуры файла-макета можно поступать одним из следующих образом:

• либо, используя команду CREATE MAKETITG, вручную создать структуру этого файла с полями FIELD_NAME, FIELD_TYPE, FIELD_LEN, FIELD_DEC;

• либо командой COPY TO MAKETITG STRUCTURE EXTENDED перенести структуру любого какого угодно файла базы данных в MAKETITG.DBF, а затем очистить этот файл от данных командой ZAP. Это способ, конечно, проще.

Далее, например, с помощью команды BROWSE файл MAKETITG.DBF заполняется характеристиками двух обязательных полей файла — поля TAB и поля ITOG:

Record #	FIELD_NAME	FIELD_TYPE	FIELD_LEN	FIELD_DEC
1	tab	n	3	0
2	itog	n	7	2

Подготовительные действия закончены. Можно приступить к программированию. Программа формирования структуры файла ITOGBRIG.DBF, его заполнения и печати таблицы приведена ниже. Здесь сначала открывается файл MAKETITG.DBF и проверяется наличие записей, описывающих поля бригад, которые могли остаться от предыдущего счета. Если записей больше двух, то помечаются и удаляются все записи, начиная с третьей. Остаются только две записи, описывающие реквизиты обязательных полей TAB и ITOG.

Далее проверяем наличие на диске бригадных файлов. Такие файлы в своем названии имеют постоянное ядро BRIG плюс номер бригады. Последовательно перебираем все возможные названия бригад (считаем, что не бывает больше 20 бригад) в переменной BR. Если находится очередной бригадный файл (IF FILE(br+'.dbf')), в файл-макет добавляется запись, где в поле FIELD_NAME заносится имя бригадного файла без расширения DBF, а в остальные поля — другие его реквизиты (тип, длина, дробные разряды). После выполнения этой части программы файл-макет должен получить следующее содержание.

Record#	FIELD_NAME	FIELD_TYPE	FIELD_LEN	FIELD_DEC
1	tab	n	3	0
2	itog	n	7	2
3	brig1	n	7	2
4	brig4	n	7	2
5	brig12	n	7	2

Затем на его основе по команде

```
CREATE itogbrig.dbf FROM maketitg.dbf
```

создается структура файла ITOGBRIG.DBF в таком виде, как она приведена выше.

Следующий этап — заполнение файла ITOGBRIG.DBF. Здесь в цикле FOR перебираются все бригадные файлы и поочередно открываются в области В (файл ITOGBRIG.DBF по умолчанию находится в области А). В текущей записи бригадного файла берется поле TAB и в файле ITOGBRIG.DBF разыскивается запись с таким же значением ее поля TAB. Если поиск неудачен, в ITOGBRIG добавляется запись, где в поле TAB заносится значение из такого же поля

бригадного файла. Затем в поле, соответствующем анализируемой бригаде, заносится выработка, а в поле ИТОГ она суммируется с накопленной суммой. Если поиск оказался успешным (т.е. такой работник ранее уже встречался и был занесен в базу ИТОГБРИГ.DBF), то заносится только выработка и осуществляется суммирование.

Завершается программа печатью полученного файла ИТИБРИГ.DBF. Одновременно в массиве S все столбцы суммируются и печатаются в последней строке документа. Результат печати показан в самом начале примера при постановке задачи.

```
*-----Программа создания, заполнения и печати файла ИТОГБРИГ.DBF-----
SET TALK OFF
SET DELETED ON
SET SAFETY OFF
CLEAR
USE maketitg
IF RECCOUNT(>>2      && Если в файле MAKETITG.DBF больше двух
    GO 3              && записей, они уничтожаются, начиная с третьей
    DELETE REST
    PACK
ENDIF
```

```
*-----Создание структуры файла ИТИБРИГ.DBF-----
kbrig=0      && Счетчик фактического количества бригад
FOR i=1 TO 20      && Перебор всех бригадных файлов
    BR='brig'+LTRIM(STR(i))
    IF FILE(BR+'.dbf')      && Если бригада с таким номером есть,
        kbrig=kbrig+1
        APPEND BLANK      && то добавляется одна строка-описатель
                           && записи в файл-макет MAKETITG.DBF
    REPLACE field_name WITH BR, field_type WITH 'n';
    field_len WITH 7, field_dec WITH 2
ENDIF
```

```
ENDFOR
CREATE itogbrig FROM maketitg      && Создание из файла-макета
                                   && файла ИТОГБРИГ.DBF
```

```
*-----Заполнение файла ИТОГБРИГ.DBF-----
FOR i=1 TO 20      && Перебор бригадных файлов
    BR='brig'+LTRIM(STR(i))
    IF FILE(BR+'.dbf'))      && Если файл с таким номером есть,
        SELECT b
        USE &br      && он открывается в области В
        SCAN      && и сканируются все его записи
            SELECT a
            * В ИТОГ.DBF ищется запись, в которой содержание поля TAB
            * совпадает с полем TAB из бригадного файла
            LOCATE FOR b.tab=a.tab
            IF !FOUND()
                * Если не найдено, ИТОГ.DBF дополняется новой записью
                * с полем ИТОГ из бригадного файла
                APPEND BLANK
                REPLACE a.tab WITH b.tab
            ENDIF
            * Заносится выработка из бригадного файла и
            * подсуммируется в поле ИТОГ
            REPLACE a.&br WITH b.vir, a.itog WITH a.itog+b.vir
        ENDSCAN
ENDIF
```

```
ENDFOR
*-----Печать файла ИТОГБРИГ.DBF-----
SELECT a
rr=kbrig*9      && Определение общей длины колонок, занимаемых
                 && столбцами бригад (по 9 на бригаду)
```

```
? PADC('ИТОГ',15+rr)
? REPLICATE('-',15+rr)
? ' ',PADC('Бригады',rr)
? 'ТАБЕЛЬ',REPLICATE('-',rr),'ИТОГО'
?
```

```
FOR i=3 TO kbrig+2      && Просмотр полей бригад
    * Выделение номера бригады из имени поля и его печать
    ?? TRANSFORM(SUBSTR(FIELD(i),5,2),'##'),
```

```
ENDFOR
? REPLICATE('-',15+rr)
DIMENSION s(kbrig)      && Массив верикальных сумм по бригадам
```



```

s=0
ss=0
SCAN
    ? ' ',TAB,
    FOR i=3 TO kbrig+2
        f=FIELD(i)
        ?? &f,
        s(i-2)=s(i-2)+&f
    ENDFOR
    ss=ss+itog
    ?? ' ',itog
ENDSCAN
? REPLICATE('-',15+rrr)
? 'ВСЕГО: '
FOR i=1 TO kbrig
    ?? TRANSFORM(s(i),'####.##'),
ENDFOR
?? TRANSFORM(ss,'####.##')
?
*-----Конец программы -----

```

&& Переменная сумм в столбце ИТОГО

&& Печать строки выработок

&& Печать ИТОГА строки

&& Печать элементов строки ВСЕГО

&& Печать конечной суммы в столбце ИТОГО

Замечание. Файл-макет не должен содержать записей, помеченных на удаление, даже если введена команда SET DELETE ON, поскольку по команде CREATE ... FROM создает новый файл из всех его записей. Ввиду этого после пометки записей в файле MAKETITG.DBF он сразу упаковывается (PACK).

24.3. Взаимодействие файлов

Кроме команды SET RELATION TO, синхронизирующей движение указателей записей в двух или более файлах, имеются команды более глубокой увязки файлов баз данных.

Слияние файлов БД. Следующая команда дает возможность создавать новую базу данных, которая является соединением двух имеющихся баз или их частей, т.е. она осуществляет слияние "по горизонтали":

■ JOIN WITH <область> TO <имя нового файла>
[FIELDS <список полей>] FOR <условие>

По команде образуется новый файл из полей активной базы данных и другой базы из указанной рабочей области. Естественно, чтобы объединение не было механическим, поля из разных файлов должны описывать один и тот же объект в каждой записи. Для этого в команду включено FOR-условие, позволяющее осуществлять нужный анализ данных для копирования.

Пример. Пусть требуется из файлов KADR.DBF и BRIG1.DBF создать третий файл, который будет содержать поля FAM (фамилия) и TAB (табель) из KADR.DBF и поле VIR (выработка) из BRIG1.DBF. Новый файл назовем KADR VIR.DBF.

```

.USE kadr IN a
.SELECT b
.USE brig1
JOIN WITH a TO kadrvir FOR tab=a.tab FIELDS a.fam, tab,vir

```

Файл KADR.DBF открыт в области А, а BRIG1.DBF — в области В. По команде JOIN соединяются нужные поля из обоих файлов в файле KADR VIR.DBF, причем очередная запись нового файла будет составляться из указанных полей только в случае, если в записях обоих файлов-источников совпадают табельные номера (FOR TAB=A.TAB).

```

.USE kadrvir
.LIST OFF
A.FAM          TAB          VIR
ПОТАПОВ Д.П.   098          330.00
КУЛАКОВА М.И.  006          400.80
СИДОРОВ П.С.   013          12.00
МИРОНОВ Р.И.   468          170.00

```

Для проверки работы программы командой LIST выводится содержимое нового файла KADR VIR.DBF.

Модификация БД в зависимости от другой базы. Команда, рассматриваемая ниже, позволяет изменить (вычислить) содержимое полей одного файла базы данных в зависимости от другого:

```
■ UPDATE ON <ключевое поле> FROM <область> .
  REPLACE <поле> WITH <выражение>
  [,<поле> WITH <выражение>...] [RANDOM]
```

Команда UPDATE изменяет поле/поля активной базы данных, перечисленные перед словом WITH, на выражения, указанные после него. Такие <выражения> могут быть построены с участием полей другой базы данных из указанной рабочей области. Обновления производятся только для тех записей, для которых в обоих файлах совпадают значения ключевого поля.

Желательно, чтобы обе базы были упорядочены, т.е. отсортированы или проиндексированы по ключевому полю. Активная база может быть и не упорядочена. В этом случае в команде необходимо указать опцию RANDOM.

Например, в качестве исходной изменяемой базы возьмем KADR VIR.DBF, а — второй BRIG1.DBF. Ранее мы создали базу KADR VIR, в которой увязаны табельные номера, фамилии работников и их выработки за истекший месяц. В следующем месяце выработки членов бригады меняются, а это значит, что значение поля VIR в файле KADR VIR должно быть заменено на новое из файла BRIG1.DBF. Это можно осуществить следующим образом:

```
.USE brig1 IN a
.INDEX ON tab TO brig1tab
.SELECT b
.USE kadrvir
.INDEX ON tab TO kvirtab
.UPDATE ON tab FROM a REPLACE vir WITH a.vir
```

Корректность выполнения команды обеспечивается созданием двух индексных файлов по полю TAB — для файлов KADR VIR. DBF (индекс KDVIRTAB.IDX) и BRIG1.DBF (индекс BRIG1TAB. IDX).

Файл BANK.DBF		Файл BANKDAY.DBF			Файл BANK.DBF	
FAMKL	SUMMA	FAMKL	PRIH	RASH	FAMKL	SUMMA
1 Кулик М.	200	1 Кулик М.	500		1 Кулик М.	700
2 Лукин С.	1300	2 Лукин С.	1000	200	2 Лукин С.	2100
3 Васин А.	5000				3 Васин А.	5000
4 Жуков М.	600	3 Жуков М.	7000	500	4 Жуков М.	7100
5 Петров В.	300				5 Петров В.	300
6 Савин А.	8000	4 Савин А.	1000		6 Савин А.	7000

Рис.24.3

Другой пример. Пусть имеются два файла, связанные с банковскими операциями. Один файл BANK.DBF содержит поля с фамилиями клиентов и суммами на их счетах (поля FAMKL и SUMMA), другой файл BANKDAY.DBF — информацию об операциях за истекший день. Последний имеет три поля: фамилий (FAMKL), прихода (PRIH) и расхода (RASH).

Составим программу обновления поля SUMMA в соответствии с значениями прихода и расхода, т.е. SUMMA=SUMMA+PRIH-RASH:

```
.USE bankday IN a
.INDEX ON famkl TO bankfam  && Индексирование базы BANKDAY.DBF
                              && по полю фамилий (FAMKL)
.SELECT b
.USE bank
.UPDATE ON famkl FROM a REPLACE summa WITH summa+a.prih-a.rash
```

На рис.24.3 приведено содержимое файлов BANK и BANKDAY, а также новое (справа) значение файла BANK после исполнения команды UPDATE.

24.4. Сортировка данных

Наряду с индексированием базы данных, которое при неизменности файла позволяет предъявлять записи и перемещаться по нему в желаемом порядке, существует команда, выполняющая физическое упорядочение файла:

- **SORT TO** <имя нового файла> [ASCENDING/DESCENDING]
ON <поле> [/A] [/D] [/C] [, <поле> [/A] [/D] [/C]...] [<границы>]
[FOR <условие>] [WHILE <условие>] [FIELDS <список полей>]

Команда создает из активной базы данных новый файл, в котором записи расположены в возрастающем (/A) или убывающем (/D) порядке относительно указанного поля/полей. Если параметр сортировки не указан, по умолчанию подразумевается /A — возрастание. Ключ /C означает, что при сортировке будет игнорироваться регистр букв (строчные/заглавные). Для кириллицы этот механизм, естественно, работает только в русифицированных версиях FoxPro. Допускается соединение ключа C с другими ключами, например /DC.

Сортируемые поля перечисляются в команде в порядке их значимости. Может быть выполнена сложная сортировка по нескольким полям (одновременно до 10). Однако не разрешается указывать функции от полей, как это возможно при индексировании.

Опции ASCENDING/DESCENDING означают, что сортировка будет вестись по возрастанию/убыванию (ASCENDING по умолчанию). Использование слова DESCENDING по существу "переворачивает" ключи /A и /C для каждого из полей на обратный.

Новый файл образуется из всех или некоторых перечисленных полей (если есть параметр FIELDS) записей, удовлетворяющих указанным условиям и находящихся в указанных границах. По умолчанию область действия команды — весь файл. В список полей можно включать поля и из других активных баз данных.

П р и м е р. Пусть из базы KADR.DBF нужно создать новый файл KADR.SORT.DBF, содержащий поля FAM (фамилия), DET (количество детей) и DTR (дата рождения). Файл должен быть отсортирован в поле DET по убыванию количества детей (главное поле) и в поле FAM в алфавитном порядке. Последнее означает, что записи о лицах, имеющих одинаковое количество детей, располагаются в порядке возрастания (по алфавиту) их фамилий.

```
.USE kadr
.SORT TO kadr.sort ON det/D, fam FIELDS fam,det,dtr
.LIST OFF det,fam
```

Дополнение уже отсортированного файла должно выполняться так, чтобы не разрушить его порядок. После того как найдено место, куда должна быть внесена новая запись, нужно использовать команду вставки

- **INSERT [BEFORE] [BLANK]**

В зависимости от формы команды она выполняет различные действия:

INSERT — вызывает предъявление полноэкранного режима, подобного командам CHANGE и APPEND, пользуясь которым можно ввести данные в новую запись сразу за текущей;

INSERT BEFORE — то же, но вставка новой записи перед текущей;

INSERT BLANK и **INSERT BEFORE BLANK** — осуществляет вставку пустой записи в файл без каких-либо предъявлений.

Рассматриваемая команда не является инструментом работы лишь только с

отсортированными файлами, но вставка записей в физически неупорядоченный файл не имеет большого смысла.

Здесь уместно провести сравнение упорядочения сортировкой и индексированием. На практике следует отдать предпочтение индексированию по следующим причинам:

- о индексный файл создается в самом начале работы с базой данных, и в дальнейшем его обновление происходит быстро и незаметно для пользователя по мере редактирования или ввода новых записей;

- о индексный файл занимает сравнительно немного места на диске;

- о команда сортировки для баз заметных размеров выполняется медленно; то же можно сказать и о вставках записей, так как при этом происходит перемещение всех записей, следующих за вставляемой;

- о поиск в базе данных при наличии индекса идет быстрее, чем в отсортированном файле; факт физической упорядоченности записей в базе не влечет никакого выигрыша в скорости последовательного поиска.

Даст ли какие-нибудь преимущества в скорости поиска и обработки данных одновременное упорядочение индексированием и сортировкой по одному и тому же ключу? Да, такую ситуацию можно предположить, если необходим поиск не отдельной записи, а их большой группы, объединяемой общим признаком. Первую такую запись лучше разыскать командой/функцией ускоренного поиска SEEK. Поскольку следующие искомые записи в отсортированном файле находятся физически непосредственно ниже найденной, теперь целесообразно отключить более медленный в данном случае индекс и пользоваться командой SCAN с WHILE-условием.

Замечание. При сортировке следует иметь в виду, что для ее исполнения временно необходима дополнительная дисковая память, равная удвоенному размеру исходного файла.

24.5. Математическая обработка БД

В FoxPro имеются средства для получения некоторых простых числовых характеристик данных, а именно количеств, сумм, средних и некоторых других величин. Рассмотрим эти команды.

- COUNT [<границы>] [WHILE <условие>]
[FOR <условие>] [TO <переменная>]

По команде COUNT подсчитывается число записей в заданных границах, удовлетворяющих условиям, которое заносится в указанную <переменную>.

- SUM [<границы>] [WHILE <условие>]
[FOR <условие>] <список выражений>
[TO <переменные> / TO ARRAY <массив>]

По команде SUM суммируются значения перечисленных числовых полей в указанные <переменные> или <массив>. В списке выражений разрешается указывать не только имена числовых полей, но и функции от них и функции от нескольких полей одновременно. Это значит, например, что можно воспользоваться функцией STR() и просуммировать символьные поля с цифровыми данными. Можно просуммировать квадратные корни величин и т.д. Если <переменных> не было к моменту исполнения команды, то они будут созданы, однако <массив> должен уже существовать.

- AVERAGE [<границы>] [WHILE <условие>]
[FOR <условие>] <список выражений>
[TO <переменные> / TO ARRAY <массив>]

По команде AVERAGE подсчитывается среднее арифметическое при тех же допущениях, что и для предыдущей команды.

- CALCULATE [<границы>] [WHILE <условие>]
[FOR <условие>] <список выражений>
[TO <переменные>]/TO ARRAY <массив>]

Команда CALCULATE позволяет вести математические расчеты в базе данных. <Список выражений> может содержать любую комбинацию следующих внутренних для данной команды функций:

- AVG(<вырN>) — среднее арифметическое для выражения с полем;
- CNT() — число записей в базе данных;
- MAX(<выр>) — максимальное значение <выр> от поля любого типа;
- MIN(<выр>) — минимальное значение <выр> от поля любого типа;
- STD(<вырN>) — среднеквадратическое отклонение <вырN>;
- SUM(<вырN>) — суммирование значений поля или выражения с полем;
- VAR(<вырN>) — дисперсия <вырN>
- NPV(...) — финансовая функция, которую мы рассмотрим позже.

Вышеперечисленные команды по умолчанию имеют область действия весь (ALL) файл, если не указаны <границы> и/или <условия>.

П р и м е р. Из файла KADR.DBF найти следующие величины:

- количество всех мужчин, родившихся после 1950 г. (переменная K);
- количество детей у всех работников предприятия (D);
- среднее количество детей, приходящееся на каждую женщину (S);
- максимальную, среднюю и минимальную среднюю зарплату (Z1, Z2, Z3).

```
.USE kadr
.COUNT FOR pol='M'.AND.YEAR(dtr)>1950 TO k
.SUM det TO d
.AVERIGE FOR pol='Ж' det TO s
.CALCULATE MAX(szar),AVG(szar),MIN(szar) TO z1,z2,z3
```

Все эти команды соединены здесь лишь для примера. На практике это нерационально, так как для исполнения любой такой команды необходим просмотр всего файла, в данном случае четыре раза. Таким образом, если предстоит найти не одну, а несколько сводных характеристик файла, лучше не прибегать к рассмотренным командам, а запрограммировать этот процесс в цикле, где каждая запись анализируется один раз, но зато по всем интересующим параметрам.

В команде CALCULATE допускается использование финансовой функции:

NPV(<вырN1>,<вырN2>[,<вырN3>])

которая позволяет вычислять приведенную к текущему моменту итоговую прибыль с учетом первоначальных вложений (<вырN3>) и доходов/убытков (<вырN2>) за каждый регулярный период времени. Кроме того, здесь учитывается ожидаемый процент прибыли (<вырN1>) с вложенных средств, выраженный в десятичной форме.

Или, иначе,

<суммарная прибыль>=NPV(<процент прибыли>,
<ожидаемые доходы/убытки> [,<начальный вклад>])

Начальный вклад, как и все убытки (если есть), должен быть отрицательным числом.

П р и м е р. Положим, начальный вклад составил 100 000 рублей, а ожидаемые прибыли в течение следующих пяти лет должны составить соответственно 1000, 10000, 30000, 50000 и 60000. Положим также, что процент прибыли, при котором имеет смысл вкладывать средства, 12 %.

Для хранения ожидаемых значений поступлений создадим базу данных POST.DBF с полем прибыли PRIB, в которое и занесем последовательно пять вышеуказанных чисел. Тогда с помощью команд

```
USE post
CALCULATE NPV(.12,prib,-100000) TO p
```

в переменной P мы получим результат, равный — 3960.28, из которого следует, что, хотя общая прибыль в течение пяти лет составит 151 000 рублей, она все равно не покроет понесенных расходов с учетом минимально ожидаемой прибыли 12 % годовых. Следовательно, от такого

капиталовложения следует отказаться или удовлетвориться меньшим процентом прибыли. Инвестиции можно совершать, только если $NPV() \geq 0$.

Функция $NPV()$ получает результат последовательным суммированием всех полей $PRIB$, деленных на величину $(1 + \text{процент})$ в степени, соответствующей номеру периода, а затем вычитает из него первоначальный вклад. То есть тот же результат можно получить и с помощью функции $SUM()$ следующего вида:

```
CALCULATE SUM(prib/(1+.12)**RECNO()) TO p
p=p-100000
```

но гораздо медленнее.

```
■ TOTAL ON <выражения> [<границы>]
  [WHILE <условие>] [FOR <условие>]
  [FIELDS <имена полей>] [TO <файл>]
```

По команде **TOTAL** создается новый файл, куда последовательно заносятся суммы числовых полей для тех записей, которые имеют одинаковое значение <выражения>. Все такие записи исходного файла образуют одну запись в новом файле, где числовые поля будут суммами числовых полей записей исходного файла, а остальные поля равны значениям полей первой по порядку записи с заданным ключом.

Исходный активный файл должен быть отсортирован или проиндексирован по заданному ключу. Структура нового файла будет идентична структуре исходного файла. Список полей в команде может содержать имена только числовых полей, по которым выполняется суммирование. Отсутствие списка означает суммирование для всех числовых полей файла.

Эта мощная команда — эффективное средство для сбора и извлечения сведений из базы данных.

Ввиду того что среди баз, рассмотренных до сих пор, отсутствует удобный объект для применения команды **TOTAL**, создадим его сейчас.

П р и м е р. Вспомним, что ранее бригадные выработки сохранялись в файлах **BRIG1**, **BRIG2** и т. д. и что одни и те же люди в течение месяца могут работать некоторое время в составе разных бригад. Очевидно, естественным является желание получить файл, где каждый человек (табельный номер) встречался бы только один раз, но в поле выработки (**VIR**) имел бы суммарное значение, собранное из всех бригад.

Сделаем это следующим образом. Сначала объединим все бригадные файлы в один с именем **BRIGSUM.DBF** с индексным файлом по полю **TAB** (табель) **BTAB.IDX**. Пусть файл **BRIGSUM.DBF** имеет структуру, идентичную структуре бригадных файлов. Затем по команде **TOTAL** сформируем новый файл **BRIGNOV.DBF**, в котором выработка каждого рабочего уже представлена только одной записью, где в поле **VIR** (выработка) будет сумма всех его выработок во всех бригадах.

```
USE brigsum INDEX btab
ZAP
FOR i=1 TO 20
  p='BRIG'+LTRIM(STR(i,2))+'.DBF'
  IF FILE (p)
    APPEND FROM &p
  ENDIF
  i=i+1
ENDDO
TOTAL ON TAB TO brignov FIELDS vir
```

Здесь открывается и очищается файл **BRIGSUM.DBF** совместно с индексным файлом **BTAB.IDX**. Далее, поскольку количество бригад может измениться (положим только, что их не может быть более 20) и бригады могут быть пронумерованы не по порядку, последовательно проверяется наличие на диске файлов от **BRIG1.DBF** до **BRIG20.DBF**. Если очередной такой файл найден, он присоединяется к файлу **BRIGSUM.DBF** по команде **APPEND FROM &P**. В конце командой **TOTAL** формируется суммарный файл **BRIGNOV.DBF** всех выработок.

24.6. Работа с мемо-полями

В СУБД FoxPro имеется несколько команд, специально предназначенных для работы с мемо-полями базы данных.

■ APPEND MEMO <мемо-поле> FROM <файл> [OVERWRITE]

По команде APPEND MEMO добавляются данные из <файла> в <мемо-поле>. Имя файла должно быть задано с расширением. При использовании параметра OVERWRITE данные в мемо-поле будут перекрыты содержимым <файла>.

■ COPY MEMO <мемо-поле> TO <файл> [ADDITIVE]

По команде COPY MEMO копируется мемо-поле во вновь создаваемый текстовый <файл>. По умолчанию <файл> получит расширение TXT. Если задан режим ADDITIVE, мемо-поле будет добавлено в конец существующего текстового файла.

■ MODIFY MEMO <мемо-поле1>[,<мемо-поле2>...] [NOEDIT] [NOWAIT] [RANGE <вырN1>,<вырN2>] [SAVE] [WINDOW <окно>]

По команде MODIFY MEMO открываются окна редактирования мемо-полей в любой из рабочих областей. Выход из окна с сохранением изменений осуществляется нажатием клавиш Ctrl-W/End, без сохранения — клавиш Ctrl-Q или Escape.

Опции команды:

NOEDIT — допускается только просмотр поля.

NOWAIT — открытие окна не прерывает программу, которая продолжает выполняться с команды, следующей за командой MODIFY MEMO. Такое окно в программе обычно должно быть "подперто" командой READ.

RANGE — открывает окно с уже выделенными символами, начиная с позиции <вырN1> до позиции (не включая) <вырN2>. Если <вырN1>=<вырN2>, символы не выделяются, а курсор устанавливается в позицию <вырN1>. Здесь могут быть использованы функции AT() и ATC() для поиска и выделения в мемо-поле нужной строки.

SAVE — сохраняет окно на экране после завершения команды.

WINDOW <окно> — мемо-поле будет доступно для редактирования в ранее определенном командой DEFINE WINDOW окне, которому могут быть переданы атрибуты редактора FoxPro (GROW, FLOAT, ZOOM и т.д.).

Если в команде перечислен список мемо-полей, то каждое из них будет открыто в своем собственном системном окне, заголовок которого будет содержать полное имя мемо-поля. Если задана опция WINDOW, все мемо-поля открываются в этом одном окне, при этом возможен доступ к ним по очереди, например при нажатии клавиш Ctrl-F1.

■ CLOSE MEMO <мемо-поле1> [,<мемо-поле2 ...>] / ALL

По команде CLOSE MEMO закрываются окна редактирования мемо-полей открытых командами MODIFY MEMO и BROWSE. Все внесенные изменения сохраняются. Мемо-поля могут быть указаны и из других областей. Опция ALL действует на все открытые окна во всех областях. Аналогичный эффект оказывает нажатие клавиш Ctrl-W/End. Нажатие Ctrl-Q или Escape закрывает окно без сохранения изменений.

■ SET WINDOW OF MEMO TO [<окно>]

Команда указывает <окно>, которое можно использовать по умолчанию для

редактирования мемо-поля в командах APPEND, BROWSE, CHANGE, EDIT, GET/READ или MODIFY MEMO. Ввод команды без параметра <окно> отменяет назначение.

П р и м е р. Пусть в базе данных KADR.DBF требуется проанализировать трудовую деятельность всех мужчин, в особенности работавших когда-либо ранее в отделе главного механика (ОГМ), и переслать сведения об этом в текстовые файлы для просмотра и редактирования. Все сведения о прошлой работе сотрудников содержатся в мемо-поле PER (Перемещения).

Программа, решающая эту задачу, приведена ниже. Здесь сначала определяется окно для вызова мемо-поля PER, в заголовке которого содержатся указания о действии клавиш Ctrl-End (переслать мемо-поле в файл) и сканируются все записи базы, где значение поля POL равно "М" (Мужчина). Для каждой записи в первой строке выводятся табельный номер и фамилия сотрудника. Затем открывается окно для просмотра мемо-поля, причем таким образом, что курсор сразу становится в позицию (если есть), с которой начинается слово ОГМ, что облегчает анализ текста. В зависимости от содержимого мемо-поля PER пользователь принимает решение о том, нужны ли ему эти данные (нажимает Ctrl-End) или нет (Escape). В первом случае (LASTKEY()-23) табельный номер превращается в строковую переменную T, а затем мемо-поле пересылается в файл, который имеет расширение TXT, и имя, совпадающее с табельным номером. Например, для работника с табельным номером 234 будет создан текстовый файл с именем 234.TXT. Это удобно для дальнейшего установления принадлежности файла определенному человеку. Затем текстовый файл вызывается на редактирование в текстовый редактор (MODIFY FILE), после чего записывается на место старого значения мемо-поля.

Созданные текстовые файлы остаются на диске для возможной дальнейшей обработки или анализа.

```
SET SAFETY OFF
USE kadr
DEFINE WINDOW per FROM 2,1 TO 10,50 TITLE;
    'Ctrl-End - переслать в файл, нет - Esc'      && Окно для мемо-поля
SCAN FOR pol='M'
    CLEAR
    @ 1,5 SAY 'Табель - '+STR(TAB,3)+' фамилия - '+fam
    MODIFY MEMO per NOEDIT WINDOW per RANGE;
        AT('ОГМ',per),AT('ОГМ',per)              && Вызов окна для просмотра
    IF LASTKEY()-23                                && Если нажаты Ctrl-End,
        t=STR(tab,3)
        COPY MEMO per TO &t                        && мемо-поле копируется в файл,
        MODIFY FILE &t                             && вызов редактора,
        APPEND MEMO per FROM &t..txt OWRITE && возврат мемо в БД
    ENDIF
ENDSCAN
```

Пример, конечно, несколько искусственный, поскольку ничто нам не мешает сразу отредактировать мемо-поле в команде MODIFY MEMO, но тогда не будут созданы текстовые файлы.

Дисковое пространство под мемо-поля отводится в соответствии с командой
■ SET BLOCKSIZE TO <вырN>

При значении <вырN>, находящемся в диапазоне 1...32, дисковая память выделяется блоками по <вырN>*512 байт. Если <вырN> больше 32 — блоками по <вырN> байт. По умолчанию SET BLOCKSIZE = 64 (т.е. 64 байта). Чем меньше размер блока, тем экономнее расходуется дисковая память, но одновременно снижается скорость доступа к данным при больших размерах мемо-полей.

Для анализа содержимого мемо-полей могут быть использованы практически все строковые функции, а также специальные мемо-функции ATLINE(), RATLINE(), MLINE(), MEMLINES() (см. гл.16). Позже будет рассмотрено еще одно, очень удобное средство доступа к мемо-полям — команда @ ... EDIT.

24.7. Вывод текстовых файлов

Команда выводит текстовый <файл1> на экран/окно, принтер (TO PRINTER) или в другой файл (TO FILE):

■ TYPE <файл1> [TO PRINTER/TO FILE <файл2>] [NUMBER]

С параметром NUMBER в начале каждой строки будет выдаваться ее номер. Эта опция удобна для вывода текстов программ.

По умолчанию выдача текста сопровождается разбивкой на страницы и прогоном страниц. Если это не нужно, следует воспользоваться командой

■ SET HEADING OFF

(по умолчанию ON). Команда TYPE является важным средством печати программ, а также текстовых файлов, созданных в прикладных системах.

В заключение необходимо указать на одну очень полезную системную числовую переменную

■ _TALLY

в которой запоминается количество записей, обработанных командами APPEND FROM, AVERAGE, CALCULATE, COPY TO, COUNT, DELETE, INDEX, JOIN, PACK, REINDEX, REPLACE, SELECT — SQL, SORT, SUM, TOTAL, UPDATE. При старте FoxPro переменная _TALLY содержит 0.

Глава 25. СРЕДСТВА УПРАВЛЕНИЯ В СТИЛЕ WINDOWS

В FoxPro появилась группа команд вида @...GET, которые позволяют создать совершенно новые объекты пользовательского интерфейса. С их помощью можно реализовать средства управления и доступа к данным, аналогичные по форме тем, которые применяются в среде Microsoft Windows (а еще раньше на компьютере Macintosh). Новый интерфейс реализован в системных диалоговых окнах самой оболочки FoxPro, начиная с версии 1.0. В версии 2.0 эти средства уже доступны и для программирования.

Введение команд в стиле Windows является предпосылкой к созданию разновидности пакета FoxPro, целиком работающего в этой модной графической среде. Известно, что на подходе новая версия FoxPro-2.5 for Windows.

Все объекты в стиле Windows (далее просто GET-объекты) активируются командой READ, что дает возможность органично соединять GET-объекты всех видов между собой и с окнами редактирования, реализованными обычными командами GET, создавая гибкий, удобный и насыщенный пользовательский интерфейс, сочетающий сразу многие виды управления и доступа к данным. Это позволяет проектировать очень компактный интерфейс, когда все средства управления находятся "под рукой" пользователя и он во многом освобожден от "блуждания" по меню, находящихся в разных экранах и на разных уровнях прикладной системы.

Все GET-объекты чувствительны к мыши, и именно при работе с ней они наиболее эффективны. Работа без мыши хотя и доступна, но неудобна, поскольку делается невозможным быстрое перемещение между объектами.

Ниже приведены перечень новых GET-объектов и перевод их названий. Хотя эти понятия вполне утвердились на Западе для Windows-подобных интерфейсов, в русском языке пока не имеется соответствующих терминов. Ввиду этого здесь дан скорее не перевод, а толкование в соответствии с внешним видом перечисляемых объектов и выполняемыми функциями, вместо которого читатель, возможно, захочет ввести свое.

Check Boxes	— кнопки-переключатели (имеют только два состояния: включено-выключено);
Invisible Buttons	— невидимые кнопки;
Lists	— списки-меню;
Popups	— скрытые POPUP-меню. Этот объект можно также назвать "кнопка-меню";
Push Buttons	— триггерные (фиксирующиеся) кнопки;
Radio Buttons	— радио-кнопки; называются так, потому что их действие похоже на действие кнопок настройки автомобильного радиоприемника, где из всех кнопок может быть нажата только одна;
Text Edit Regions	— области (регионы) редактирования.

Слово "кнопка" (Button), конечно, используется в фигуральном смысле. Под ней понимается некоторая выделенная область экрана, выбор которой нажатием клавиш Enter и Space может быть зафиксирован в заданной переменной и в дальнейшем обработан. Здесь имеется некоторое различие в реакции на их нажатие. Обычно клавиша Enter вызывает также автоматическое движение дальше (где это имеет смысл). Нажатие клавиши Space курсор не перемещает. Надо отметить, что в среде Windows отображение некоторых видов "кнопок" действительно очень похоже на настоящие кнопки.

Структуры команд, генерирующих перечисленные GET-объекты, в общем весьма схожи и обычно содержат следующие компоненты:

@ <X,Y> — экранные координаты (номера строки и колонки) левого верхнего угла объекта или начала строки;

GET/EDIT <переменная> — переменная, элемент массива или поле базы данных, запоминающие сделанный выбор или вызванные для редактирования;

FUNCTION/PICTURE <вырС> — эти опции определяют содержание и вид элемента/элементов объекта. Действие слов FUNCTION/PICTURE совершенно одинаково, но в <вырС> со словом PICTURE первым должен использоваться знак "@".

Поскольку все такие команды очень похожи, для того чтобы их можно было отличить друг от друга, в самое начало <вырС> (или после знака "@" — если используется опция PICTURE) обычно включается некоторый определенный синтаксисом команды обязательный индивидуальный символ/символы. Например, *С,*I,*& и т.д. <ВырС> обычно имеет ключи, стоящие сразу за обязательными символами и означающие следующие действия:

N — команда не завершает команду READ (обычно принято по умолчанию);

T — команда завершает действие команды READ;

H — элементы располагаются горизонтально;

V — элементы располагаются вертикально (обычно действует по умолчанию).

Ключи могут сочетаться (например, TH, NV). Далее мы не будем больше возвращаться к разъяснению этих ключей, а только их перечислим (если есть) для каждой из команд, а также укажем действие по умолчанию (если оно изменено).

Если <вырС> может содержать слова-приглашения, то разрешается устанавливать в них "горячие" клавиши, т.е. буквы, нажатие которых на клавиатуре вызывает выбор этого элемента. Таким буквам должны предшествовать тогда символы "<".

В списке приглашений можно указывать и неактивные элементы (они пропускаются курсором). Такие элементы должны начинаться с символов "\". В списках меню для подавления доступа к строкам меню достаточно одного знака "\".

DEFAULT <выр> — задает исходное значение <переменной> равным <выражению>. Если она уже существует, DEFAULT игнорируется.

VALID <вырL> — условие, проверяемое после выбора/изменения данных.

WHEN <вырL> — условие, определяющее возможность доступа к объекту. Обе последние опции включены не столько для проверки или ограничения доступа к элементам (здесь это часто бессмысленно), сколько с целью обращения к пользовательским функциям для передачи в них сделанного выбора и возможной дополнительной обработки и/или предъявления данных.

MESSAGE <вырС> — дополнительное сообщение, появляющееся в центре нижней строки экрана или в строке, определенной командой SET MESSAGE TO.

ENABLE/DISABLE — активация/деактивация объекта. По умолчанию — ENABLE.

COLOR SCHEME <вырN>/COLOR <список цветовых пар> — указание цвета номером цветовой схемы или списком цветовых пар. По умолчанию объекты "берут" цвета, принятые для текущего окна/экрана. Какие именно цветовые пары — см. в документации.

Перечисленные опции далее не разъясняются, за исключением случаев, когда их смысл отличается от указанного.

Поскольку все новые команды "родственны" рассмотренной ранее команде @...SAY...GET, существо некоторых совпадающих опций и параметров можно уточнить в ее описании.

Кнопка-переключатель (Check Boxes)

```
@ <X,Y> GET <переменная>
  FUNCTION <вырC1>/PICTURE <вырC2>
  [DEFAULT <выр>] [ENABLE/DISABLE]
  [MESSAGE <вырC3>] [VALID <вырL1>] [WHEN <вырL2>]
  [COLOR SCHEME <вырN>/COLOR <список цветовых пар>]
```

Команда создает кнопку-переключатель, которая может изменять некоторую <переменную> с значения "Ложь" (.F.) на "Истина" (.T.) или с значения 0 на 1 и наоборот ("Отжата"/"Нажата"). Такая кнопка отображается на экране символом X (если .T.) или пробелом (если .F.), взятым в квадратные скобки, с возможным разъяснением ее смысла (prompt — приглашением), указанным справа. Например:

[X] Извещение послано

Опции команды:

GET <переменная> — имя <переменной> числового или логического типа. Если исходное значение переменной не равно 0 или равно .T., будет изображено [X], в противном случае — пробел ([]). Нажатие клавиши Enter/Space, кнопки мыши или "горячей" клавиши вызовет изменение значения переменной на противоположное.

FUNCTION <вырC1>/PICTURE <вырC2> — эти опции определяют содержимое и форму строки-приглашения ("*C" — обязательный первый элемент в <вырC>), в которой могут присутствовать знаки "<" и "\"", поставленные перед приглашением. Разрешены ключи N, T.

Пример:

```
izv=.F.
@ 6,5 GET izv PICTURE '*C \<ИЗВЕЩЕНИЕ ПОСЛАНО' COLOR SCHEME 8
@ 6,35 GET opl FUNCTION '*C \<СЧЕТ ОПЛАЧЕН' DEFAULT 1
READ
```

Здесь генерируются две кнопки вида

[] ИЗВЕЩЕНИЕ ПОСЛАНО

[X] СЧЕТ ОПЛАЧЕН

"Горячими" клавишами являются буквы "И" и "С". Исходное значение переменной IZV — .F.. Переменная OPL создается прямо в команде и получает начальное значение 1. Далее они могут быть изменены пользователем (переменная IZV — на .T., OPL — на 0) и затем проанализированы в программе через опцию VALID. На практике лучше использовать какую-то единообразную технику задания переменных. Очевидно, удобнее применять логические переменные (их легче потом анализировать).

Невидимые кнопки (Invisible Buttons)

```
@ <Y,X> GET <переменная>
  FUNCTION <вырC1>/PICTURE <вырC2> [DEFAULT <вырN1>]
  [SIZE <вырN2>,<вырN3> [,<вырN4>]] [ENABLE/DISABLE]
  [MESSAGE <вырC3>] [VALID <вырL1>] [WHEN <вырL2>]
  [COLOR SCHEME <вырN5>/COLOR <список цветовых пар>]
```

Команда создает так называемые невидимые кнопки на экране/окне, которые мы можем выбрать как элементы меню. Невидимыми они называются потому, что не формируют никаких текстовых сообщений и

только выделяются цветом. В эти области с помощью команд @ ... SAY можно поместить любые строки. Невидимые кнопки используются для создания кнопок-пиктограмм, конечно, настолько, насколько это возможно в неграфическом режиме.

При нажатии клавиш Enter/Space может быть сделан вызов ПФ через опцию VALID. Выход из меню возможен по достижении последней кнопки или по команде CLEAR READ, которая может содержаться в ПФ.

После завершения выбора из такого меню <переменная>, которая должна быть числового типа, получает значение — номер выбранной кнопки и может быть обработана в программе. Такие команды дают возможность создать произвольное объемное меню, где каждый его элемент может занимать прямоугольник любого размера и положения.

Опции команды:

FUNCTION <вырС1>/PICTURE <вырС2> — здесь <вырС> должно содержать обязательные первые символы — "*I". Чтобы создать несколько кнопок одной командой, следует после символа I поставить нужное число знаков ";", минус единица. Так, четыре кнопки создаст опция FUNCTION '*I;;;'. Допускаются ключи H и V, N и T и знак "\\".

SIZE <вырN2>,<вырN3>[,<вырN4>] — эти параметры определяют высоту (вырN2), длину (вырN3) и расстояние между кнопками (вырN4). По умолчанию высота и длина равны 0, расстояние — 1.

Пример:

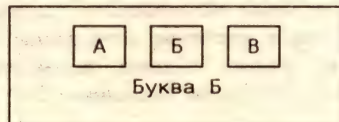
```
@ 1,2 GET k FUNCTION '*I ;\\; ;' SIZE 3,5,2 DEFAULT 1
READ
```

Здесь создается пять кнопок, расположенных вертикально, размером 3 строки на 5 колонок и расстоянием между ними 2 колонки. Вторая кнопка пропускается (\\). Исходное значение переменной K равно 1.

В следующем примере создается три кнопки, содержащие буквы А, Б, и В. При выборе каждой кнопки под ней появляется отобранная буква. Возможный экран изображен внизу справа (клавиша Enter нажата, когда курсор находится на букве Б).

```
@ 5,4 SAY 'А Б В'                                && Формирование изображений букв
@ 4,2 TO 6,6                                         && и прямоугольников под невидимые кнопки
@ 4,9 TO 6,13
@ 4,16 TO 6,20
@ 4,2 GET b FUNCTION '*IH ;;' SIZE 3,5,2 DEFAULT 3:
  VALID(bukva()) MESSAGE 'Выберите букву'
READ CYCLE
```

```
FUNCTION bukva                                     && Функция вывода отобранной буквы
@ 8,1 CLEAR
DO CASE
CASE b = 1
  @ 8,2 SAY 'Буква А'
CASE b = 2
  @ 8,9 SAY 'Буква Б'
CASE b = 3
  @ 8,17 SAY 'Буква В'
ENDCASE
RETURN
```



Пример. Пусть необходимо создать несколько видов кнопок, изображающих некоторые пиктограммы — картотечную стойку с ящиками и дверь с надписью ВЫХОД.

Само отображение этих предметов осуществляется с помощью команд TEXT...ENDTEXT. Два левых ящика картотеки выделяются первой командой @...GET, следующие два — другой, дверь выделяется последней командой такого типа. Обработка (здесь только вывод номера кнопки) любого выбора осуществляется функцией VIB(), куда передается этот номер. С тем чтобы номера были разными и шли по порядку от 1 до 5, во второй группе кнопок в

функцию VIB значение K передается увеличенным на два, а в третьей — на четыре. В приведенном ниже решении используется для обработки всех выборов только одна функция. Если для каждой группы кнопок создать свои процедуры/функции, то значение параметра уже не нужно делать уникальным.

```
CLEAR
SET TALK OFF
TEXT
```

&& Предъявление пиктограмм

КАРТОТЕКА	
Рабочие	Вспомогат. службы
Инженеры Техники	Руководство



```
ENDTEXT
@ 7,2 GET k FUNCTION '*I' SIZE 2,9,1 DEFAULT 1 VALID(vib(k))
@ 7,12 GET k FUNCTION '*I' SIZE 2,11,1 VALID(vib(k+2))
@ 1,33 GET k FUNCTION '*IT' SIZE 12,17 VALID(vib(k+4));
MESSAGE 'Выход из системы'
```

READ CYCLE

```
FUNCTION vib && Функция индикации номера кнопки (K=1-5)
PARAMETERS k
@ 22,5 SAY k
RETURN
```

Ключ T использован при описании последней кнопки с целью придания ей функции выхода из команды READ.

Списки меню (Lists)

```
■. @ <X,Y> GET <переменная>
[FUNCTION <вырC1>]/[PICTURE <вырC2>]
FROM <массив> [RANGE <вырN1>[,<вырN2>]]/POPUP <POPUP-меню>
[DEFAULT <выр>] [SIZE <вырN3>,<вырN4>] [ENABLE/DISABLE]
[MESSAGE <вырC3>] [VALID <вырL1>] [WHEN <вырL2>]
[COLOR SCHEME <вырN5>/COLOR <список цветовых пар>]
```

Команда создает меню, которое может быть образовано элементами <массива> или активирует ранее определенное (командой DEFINE POPUP) POPUP-меню. Несколько таких команд могут создавать на экране/окне сразу несколько различных меню (списки меню), между которыми легко перемещаться и совершать необходимые выборы данных.

Опции команды:

GET <перем> — переменная, элемент массива, поле базы (числового или символического типов), которые запоминают выбор, сделанный в меню. Их исходные значения определяют, какое именно меню будет активным при начале выполнения команды.

FUNCTION <вырC1>/PICTURE <вырC2> — обязательный первый символ — "&". Допускаются ключи T и N.

FROM <массив> — список строится с помощью одно- или двумерного <массива>. В последнем случае для наполнения меню берется только первый столбец массива. Меню на экране отображается вертикально.

RANGE <вырN1>[,<вырN2>] — определяет номер начального элемента

массива (вырN1) и количество используемых элементов (вырN2).
POPUP <POPUP-меню> — создает список, используя POPUP-меню с указанным именем. Такое меню должно быть ранее описано командой **DEFINE POPUP**.

SIZE <вырN3>,<вырN4> — размер области экрана,отводимой под меню.

П р и м е р. Ниже вызвано POPUP-меню BD, которое предьявляет названия всех файлов баз данных с именем, начинающимся на BRIG.

```
DEFINE POPUP bd PROMPT FILES LIKE brig*.dbf MARGIN
@ 2,2 GET k POPUP bd SIZE 8,20 DEFAULT 1
READ
```

Очевидно, что это меню можно было бы активировать и уже известной нам командой **ACTIVATE POPUP**, а не **READ**. Однако использование такой технологии позволяет легко интегрировать POPUP-меню в единый интерфейс вместе с окнами и другими средствами управления, в том числе и с другими POPUP-меню.

При предьявлении POPUP-меню его координаты и размер определяются не командой **DEFINE POPUP**, а параметрами, заданными в команде **@...GET**.

Другая задача. Пусть для каждого работника, включенного в базу KADR.DBF, нужно найти его выработку в бригадах 1 и 2, а также установить ему процент премии, который может иметь следующие фиксированные значения: 0, 10, 30 и 50 %. Для этого нужно предьявить четыре меню: из полей FAM и TAB базы KADR. DBF, из полей TAB и VIR базы BRIG1.DBF, такое же меню из BRIG2.DBF и процентов премий. Первые три меню объявляются как POPUP-меню, а третье создается из массива PREM(4).

Фамилия/табель выбираются из базы KADR.DBF произвольно, но выработки из бригадных файлов должны отбираться только для того табельного номера, который уже был указан в предыдущих меню. Если это условие не выполняется, курсор автоматически возвращается на предыдущее меню с помощью системной переменной **_CUROBJ** (см. описание команды **READ**). Кроме того, ниже меню выводятся отбираемые данные. Все выбранные в меню строки помечаются знаком "+".

Завершение отбора осуществляется через триггерные кнопки (<Конец>/<Выход>), которые в процедуре **KON()** инициируют вывод полной зарплаты работника — сумма выработок в бригадах плюс премия, если табельные номера во всех меню совпадают.

Ниже приведена программа, реализующая эту задачу, а также изображен экран (рис.25.1) со всеми меню и уже сделанным отбором для ПОТАПОВА Д.П.

```
*-----Программа отбора данных из меню-----
CLEAR
SET TALK OFF
USE kadr IN a                && Открытие баз данных
USE brig1 IN b
USE brig2 IN c
DIMENSION prem(4)           && Создание массива
prem(1)=0
prem(2)=10
prem(3)=30
prem(4)=50
sum=
p=1
*-----Создание POPUP-меню -----
DEFINE POPUP fam FROM 0,0 PROMPT FIELDS LEFT(fam,13)+
'|'+STR(TAB,3) MARGIN TITLE 'Фамилия Таб' MARK '+'
DEFINE POPUP br1 FROM 0,0 PROMPT FIELDS STR(b.tab,3)+
'|'+STR(b.vir,8,2) MARGIN TITLE 'Бригада 1' MARK '+'
DEFINE POPUP br2 FROM 0,0 PROMPT FIELDS STR(c.tab,3)+
'|'+STR(c.vir,8,2) MARGIN TITLE 'Бригада 2' MARK '+'
*-----Отображение списка меню -----
@ 2,3 GET kdr POPUP fam SIZE 8,22 DEFAULT 1;
VALID(kdr(a.fam,a.tab)) MESSAGE 'Отберите фамилию'
@ 2,26 GET bg1 POPUP br1 SIZE 8,15 DEFAULT 1;
VALID(bg1(b.vir)) MESSAGE 'Найдите выбранный табель'
@ 2,42 GET bg2 POPUP br2 SIZE 8,15 DEFAULT 1;
VALID(bg2(c.vir)) MESSAGE 'Найдите выбранный табель'
@ 2,59 GET p FROM prem SIZE 6,5 DEFAULT 1;
VALID(premir()) MESSAGE 'Укажите процент премии'
```



```

@ 14,25 GET l FUNCTION 'Н Конец;Выход' VALID kon() DEFAULT 2
READ CYCLE                && Активация всех меню и кнопок

FUNCTION kdr               &&-----Функция обработки выбора из меню KDR
PARAMETER f,t
@ 11,4 SAY f               && Вывод фамилии и табельного номера
@ 12,4 SAY '(табель: '+STR(t,3)+' )'
RETURN

FUNCTION bg1               &&-----Функция обработки выбора из меню BG1
PARAMETER v1
IF a.tab=b.tab            && Проверка совпадения табельных номеров
@ 11,24 SAY v1            && Вывод выработки из базы BRIG1.DBF
ELSE
WAIT 'Неверный выбор ' WINDOW NOWAIT
CUROBJ=1                 && Возврат на предыдущее меню
ENDIF
RETURN

FUNCTION bg2               &&-----Функция обработки выбора из меню BG2
PARAMETER v2
IF a.tab=b.tab.AND.a.tab=c.tab
@ 11,37 SAY v2            && Вывод выработки из базы BRIG2.DBF
ELSE
WAIT 'Неверный выбор ' WINDOW NOWAIT
CUROBJ=2                 && Возврат на предыдущее меню
ENDIF
RETURN

FUNCTION premir            &&----Функция обработки выбора из массива PREM
@ 11,53 SAY 'премия: '+STR(prem(p),2)+'%'
RETURN

FUNCTION kon               &&-----Функция выхода
IF l=1.AND.a.tab=b.tab.AND.a.tab=c.tab
sum=STR((b.vir+c.vir)*prem(p)/100+b.vir+c.vir ,8,2)
WAIT 'Всего: '+sum WINDOW
ENDIF
RETURN
*-----Конец программы-----

```

Фамилия		Таб	Бригада 1		Бригада 2		
СИДОРОВ П.С.	13	3	330.00	6	400.00	0	
+ПОТАПОВ Д.П.	98	6	400.00	24	800.00	10	
КУЛАКОВА М.И.	6	13	120.00	468	170.00	30	
ПОПОВ А.А.	234	468	170.00	+ 98	800.00	50	
РОМАНОВА М.С.	890	+ 98	1000.00	16	320.00		
МИРОНОВ Р.И.	468	18	200.00	13	2000.00		
ПОТАПОВ Д.П. (табель: 98)		1000.00		800.00		премия: 50%	
<div style="float: right; border: 1px solid black; padding: 2px; display: inline-block;">Всего: 2700.00</div> <div style="clear: both;"></div>							
<div style="display: flex; justify-content: space-around;"> <Конец> <Выход> </div> <div style="text-align: center;">Укажите процент премии</div>							

Рис.25.1

Пример множественного отбора. Пусть требуется вывести меню из всех чисел текущего месяца, причем рядом с каждым числом должен стоять его день недели. Все воскресенья и субботы также должны быть показаны, но не должны быть доступны. Из меню разрешается делать выбор любого количества дней, но только по одному разу, и отобранные строки не должны быть более доступны. После завершения отбора нажатием клавиши Escape все отобранные дни должны быть отсортированы в возрастающем порядке и предъявлены на экране.

Программа, реализующая данную задачу, и само меню показаны ниже. В ее начале в переменной KD вычисляется количество дней в текущем месяце. Для этого к текущей дате прибавляется один месяц и из результата снова вычитается текущая дата. Затем объявляются три массива: A(KD) — для

формирования строк меню, C(KD) — для сохранения отбора и B(7) — со всеми названиями дней недели. Переменная L резервируется для указания количества отображенных элементов. Далее в переменной ND формируется дата, соответствующая первому числу текущего месяца.

В цикле FOR формируются все KD строк меню. Для всех чисел месяца от 1 до KD сначала в переменной X определяется номер дня недели. Затем в очередном элементе массива A формируется его наполнение. Если очередной день суббота или воскресенье, то в начале строки ставится символ "\", что делает ее хотя и видимой, но недоступной для отбора (IIF(x=1.OR.x=7,'\\',' ')). Далее в строку вносятся очередное число месяца (STR(i,2)) и название дня недели (B(X)). Для того чтобы пользователь ориентировался в списке дат, текущая дата помечена звездочкой (IIF(i=DAY(DATE()),'*',' ')). Последняя команда в цикле (ND=ND+1) осуществляет приращение числа месяца.

Собственно меню генерируется командой @...GET. Причем в начальный момент курсор будет находиться на сегодняшней дате (DEFAULT DAY(DATE())). Обработка выбора производится в функции FFF(). Здесь по команде WAIT индицируется отображенная строка меню и пересылается в очередной L-й элемент массива A. Одновременно отображенный элемент массива A получает слева символ "\", который делает его недоступным для повторного выбора, и справа — символ "+", подчеркивающий факт уже совершенного отбора.

Завершение работы с меню может быть осуществлено нажатием клавиши Escape. При этом будут выполнены команды, следующие за командой READ, — сортировка отображенных элементов (=ASORT(c,1,l)) и их вывод на экран.

```
SET TALK OFF
CLEAR
kd=GOMONTH(DATE(),1)-DATE()
DIMENSION a(kd),b(7),c(kd)
l=0
b(1)='Воскр. '
b(2)='Понед. '
b(3)='Вторник '
b(4)='Среда '
b(5)='Четверг '
b(6)='Пятница '
b(7)='Суббота '
nd=CTOD('01.'+RIGHT(DTOC(DATE()),5))
FOR i=1 TO kd
  x=DOW(nd)
  a(i)=IIF(x=1 OR x=7,'\\','')+STR(i,2)+' '
  +b(x)+IIF(i=DAY(DATE()),'*',' ')
  nd=nd+1
ENDFOR
@ 1,7 GET k FROM a DEFAULT DAY(DATE()) VALID fff()
READ
@ 1,30 SAY 'Отобрано:'
=ASORT(c,1,l)
FOR i=1 TO l
  @ 1+i,30 SAY c(i)
ENDFOR
```

		Отобрано:
1	Пятница	6 Среда
2	Суббота	8 Пятница
3	Воскр.	12 Вторник
4	Понед. *	14 Четверг
5	Вторник	19 Вторник
6	Среда +	
7	Четверг	
8	Пятница +	

```
FUNCTION fff
WAIT LEFT(a(k),10) WINDOW NOWAIT && Вывод текущего выбора
l=l+1
c(l)=LEFT(a(k),10) && Сохранение отображенных элементов
a(k)='\\'+a(k)+'+' && Блокирование в меню отображенных элементов
RETURN .f. && Предотвращение выхода из меню
```

Пример создания меню из имен файлов, предварительно занесенных в специальный массив. Здесь удобно использовать функцию ADIR(). Так, пусть требуется создать меню из имен файлов баз данных, начинающихся на "BRIG" таким образом, чтобы в нем, однако, были отображены не собственно их имена латинскими буквами, а содержание этих файлов, т.е. "Бригада 1", "Бригада 2" и т.д. Следующие команды реализуют эту задачу:


```
=ADIR(bd,'k*.dbf')      && Занесение в массив BD данных о файлах
FOR i=1 TO m             && Замещение имен файлов
    bd(i,1)='Бригада '+SUBSTR(bd(i,1),5,1)
ENDFOR
@ 3,4 GET 1 FROM bd DEFAULT 1 VALID vibor()
READ
```

Собственно обработка выбора будет выполняться в процедуре VIBOR(), которая здесь не рассматривается.

Скрытое POPUP-меню (Popups)

```
■ @ <X,Y> GET <переменная> FUNCTION <вырC1>/PICTURE <вырC2>
    [DEFAULT <выр>] [FROM <массив>]
    [RANGE <вырN1>[,<вырN2>]] [SIZE <вырN3>, <вырN4>]
    [ENABLE/DISABLE] [MESSAGE <вырC3>]
    [VALID <вырL1>] [WHEN <вырL2>]
    [COLOR SCHEME <вырN5>/COLOR <список цветовых пар>]
```

По команде создается компактное POPUP-меню (кнопка-меню). Такое меню на экране представлено только одним пунктом, заключенным в прямоугольник:

<Пункт меню>

Целиком меню раскрывается если на него установлен курсор и нажата клавиша Enter/Space. Меню удобно тем, что не загромождает экран. Это бывает важно при недостатке места.

Опции команды:

GET <переменная> — переменная (числового или символьного типов) запоминает выбор из меню. Если она символьного типа — в ней сохраняется строка-приглашение, если числового — номер выбранного элемента меню среди других. Начальное значение переменной определяет, каким именно будет исходное приглашение меню. Если переменная числовая — приглашение будет отображать пункт меню с указанным номером. Если переменная символьная — она и будет отображена. Если такого элемента нет среди пунктов меню — он будет добавлен в его конец.

FUNCTION <вырC1>/PICTURE <вырC2>

"^" — обязательный начальный символ, за которым следуют элементы меню, разделенные символом ";". Допускаются ключи N и T, и знаки "<" и "\".

FROM <массив> — опция создает меню из элементов <массива>. В этом случае содержимое опций FUNCTION <вырC1>/PICTURE <вырC2> игнорируется, но символ "^" все равно должен присутствовать. Массив может быть одно- или двумерным. Если он двумерный, то только элементы первого столбца участвуют в образовании POPUP-меню.

RANGE <вырN1>[,<вырN2>] — определяет начальный элемент (вырN1) массива и количество элементов (вырN2), используемых в меню.

SIZE <вырN3>[,<вырN4>] — здесь <вырN3> определяет высоту POPUP-меню (всегда равно 1) и его длину (вырN4). По умолчанию длина берется равной длине самой широкой строки меню.

Пример 1.

```
@ 4.2 GET fam FUNCTION;
    ^ \<Петров А.А.;\<Васин А.Н.;\—————;\<Лукин С.Н. DEFAULT 2
READ
```


Здесь исходным приглашением является второй пункт меню (DEFAULT 2) — фамилия Васин А.Н.. После выбора переменная FAM получит числовое значение от 1 до 4. Пункт номер три (горизонтальная линия) не выбирается. Горячими клавишами являются начальные буквы всех фамилий.

Пример 2.

fam= 'Попов А.А.

```
@4,2 GET fam FUNCTION ' ^ \<Петров А.А.;\<Васин А.Н.;\_____;\<Лукин С.Н. '
READ
```

Здесь начальным приглашением является фамилия Попов А.А.. Поскольку такой фамилии нет в списке, она добавляется в меню самой последней строкой. После выбора переменная FAM получит символьное значение, соответствующее отобранной фамилии.

Исходный и раскрытый виды POPUP-меню для обоих примеров изображены на рис.25.2.

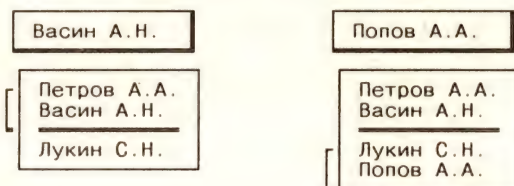


Рис.25.2

Триггерные кнопки (Push Buttons)

- @<X,Y> GET <переменная> FUNCTION <вырC1>/PICTURE <вырC2>
 [DEFAULT <выр>] [SIZE <вырN1>,<вырN2>[,<вырN3>]]
 [ENABLE/DISABLE] [MESSAGE <вырC3>]
 [VALID <вырL1>] [WHEN <вырL2>]
 [COLOR SCHEME <вырN4>/COLOR <список цветовых пар>]

По команде создаются триггерные кнопки, которые на экране изображаются в виде текста-приглашения, взятого в угловые скобки. Например, в СУБД часто встречаются кнопки

< OK > < Cancel >

Опции команды:

<X,Y> — координаты первой кнопки.

GET <переменная> — имя переменной, элемента массива или поля числового или символьного типа, запоминающих выбор пользователя. Если она числового типа, то выбор запоминается в виде номера кнопки. Если символьного — то в виде строки-приглашения из кнопки.

FUNCTION <вырC1>/PICTURE <вырC2>

"*" — обязательный первый символ, после которого может идти список приглашений для кнопок, разделенный знаком ";". Разрешаются ключи H и V, N и T (T действует по умолчанию) и знаки "<" и "\".

Одна из кнопок может быть назначена кнопкой по умолчанию. Она выделяется двойными угловыми скобками (на русифицированной клавиатуре, возможно, символами "о" и "п"). Немедленный переход к этой кнопке возможен от любой другой кнопки нажатием клавиш Ctrl-Enter. При этом сразу осуществляется ее выбор (без нажатия клавиш Enter/Space). Такая кнопка создается с помощью символов "!\". Одна из кнопок может быть назначена Escape-кнопкой (т.е. кнопкой, реагирующей на нажатие клавиши Esc). Для этого используются символы "\?".

SIZE<вырN1>,<вырN2>[,<вырN3>] — здесь <вырN1> и <вырN2> указывают высоту и длину кнопки (вырN1 всегда равно 1), а <вырN3> — промежуток между ними. По умолчанию длина кнопки определяется длиной содержащегося в ней приглашения, даже если назначена меньшая длина.

Пр и м е р ы. Следующий пример генерирует триггерное меню вида
<< ВЫХОД >> <ВОЗВРАТ К ОБРАБОТКЕ> <ПЕЧАТЬ> <ОТКАЗ>

Здесь кнопки ВЫХОД и ОТКАЗ являются соответственно кнопками по умолчанию и Escape-кнопкой.

```
i=2
@ 2,2 GET i FUNCTION;
' *N \!ВЫХОД;ВОЗВРАТ К ОБРАБОТКЕ;ПЕЧАТЬ;\?ОТКАЗ' SIZE 1,8,2
READ
```

Программа, приведенная ниже, управляет перемещением указателя записей в базе KADR.DBF. Для редактирования выведено поле FAM (Фамилия). Вертикальное меню из кнопок изображено внизу справа. Переменная, воспринимающая выбор пользователя (VIB), передается в функцию перемещения (PEREM) и изменяет положение указателя записей. Кроме того, каждое перемещение сопровождается сообщением о номере текущей записи.

```
USE kadr
@ 4,2 GET vib FUNCTION;
' *N Следующая запись;Предыдущая;Начало;Конец;Выход' VALID perem() DEFAULT 1
@ 2,2
@ 2,2 SAY 'Фамилия' GET fam
READ CYCLE
```

FUNCTION perem && Функция перемещения указателя записей

```
DO CASE
CASE vib = 1
SKIP
CASE vib = 2
SKIP -1
CASE vib = 3
GO TOP
CASE vib = 4
GO BOTTOM
CASE vib = 5
CLEAR READ
```

Фамилия ВАСИН А.Н.

<Следующая запись>
<Предыдущая>
<Начало>
<Конец>
<Выход>
Запись номер 18

```
ENDCASE
SHOW GETS                      && Обновление поля FAM
@ 9,3 SAY 'Запись номер' +LTRIM(STR(RECNO()))
RETURN
```

Здесь для сокращения примера процесс перемещения в базе упрощен. На практике следует контролировать и не допускать попыток перемещения указателя записей за первую/последнюю запись базы..

Радиокнопки (Radio Buttons)

■ @ <X,Y> GET <переменная> FUNCTION <expC1>/PICTURE <вырC2>
[DEFAULT <выр>] [SIZE <вырN1>,<вырN2>[,<вырN3>]]
[ENABLE/DISABLE] [MESSAGE <вырC3>]
[VALID <вырL1>] [WHEN <вырL2>]
[COLOR SCHEME <вырN4>/COLOR <список цветовых пар>]

По команде создаются так называемые радиокнопки. Среди таких кнопок только одна может быть нажата. Каждая кнопка представляет собой одну позицию экрана,ограниченную круглыми скобками. Если в скобках точка — значит, кнопка нажата. Если скобки пустые, то нет. Справа от скобок может быть выведена строка-приглашение. Например, следующую группу кнопок вы можете видеть в меню настройки текстового редактора FoxPro

(.) Left justify
() Right justify

() Centers justify

Опции команды:

<X,Y> — координаты первой кнопки.

GET <переменная> — переменная числового или символьного типа, в которой сохраняется сделанный выбор (номер кнопки или приглашение соответственно). Исходное значение переменной определяет исходное положение курсора. Он будет находиться на кнопке с совпадающим номером или приглашением.

FUNCTION <вырC1>/PICTURE <вырC2>

"*R" — обязательный первый элемент <вырC>, после которого идут приглашения, разделенные знаком ";". Разрешены ключи N, T, H, V и знаки "<" и "\\".

SIZE <вырN1>,<вырN2>[,<вырN3>]

здесь <вырN1> — высота кнопки (всегда вырN1=1), <вырN2> — длина, <вырN3> — расстояние между кнопками.

Пример:

```
@ 5,8 GET x FUNCTION '*RH Зарплата;Премия;Прогрессивка;Налог' DEFAULT 2
READ
```

Приведенный фрагмент программы порождает следующее множество радио-кнопок:

() Зарплата (.) Премия () Прогрессивка () Налог

Области редактирования (Text Edit Regions)

- @ <X,Y> EDIT <переменная> SIZE <вырN1>,<вырN2>[,<вырN3>]
 [ENABLE/DISABLE] [MESSAGE <вырC3>] [VALID <вырL1>
 [ERROR <вырC4>] [WHEN <вырL2>] [NOMODIFY] [SCROLL]
 [COLOR SCHEME <вырN4>/COLOR <список цветовых пар>]

Команда создает прямоугольную область на экране, в которой возможно редактирование переменной, элемента массива, поля или мемо-поля базы данных. Такой объект должен быть символьного типа. В созданном регионе действуют многие функции редактора FoxPro. Для доступа к мемо-полю не нужно нажимать клавиши Ctrl-Home. Чтобы сохранить изменения, сделанные в <переменной> следует нажать клавиши Tab/Ctrl-Tab. Выход без сохранения, как обычно, осуществляется с помощью клавиши Escape. Клавиша Enter переводит курсор на новую строку в области редактирования.

Опции команды:

SIZE <вырN1>,<вырN2>[,<вырN3>] — определяет размер области для редактирования (<вырN1> — высота области, <вырN2> — ее длина, <вырN3> — предельное число редактируемых символов).

VALID <вырL1> — условие проверяется при выходе из области редактирования клавишами Tab, Ctrl-Tab или Shift-Tab.

ERROR <вырC4> — собственное сообщение об ошибке (<вырC4>), если условие <вырL1> оказалось ложным.

WHEN <вырL2> — определяет возможность доступа к области.

NOMODIFY — редактирование запрещено (только просмотр).

SCROLL — у правой границы редактируемой области будет показан маркер положения курсора ("ползунок").

Пример. Создание области редактирования для мемо-поля PER из базы KADR.DBF размером 3 строки на 20 колонок. Координаты левого верхнего угла 10x5.

```
USE kadr
@ 10,5 EDIT per SIZE 3,20 SCROLL
READ
```


Глава 26. КОМАНДА ЧТЕНИЯ ДАННЫХ READ, МНОГООКОННЫЙ ИНТЕРФЕЙС

Команда ранее уже упоминалась и широко использовалась в примерах. Эта команда имеет огромное значение для построения пользовательского интерфейса, управляемого событиями.

Сейчас, когда рассмотрены все GET-объекты, приведем более полный ее формат, а также некоторые смежные команды:

■ READ [CYCLE] [ACTIVATE <вырL1>] [DEACTIVATE <вырL2>]
[MODAL] [WITH <список окон>] [SHOW <вырL3>]
[VALID <вырL4/вырN1>] [WHEN <вырL5>]
[OBJECT <вырN2>] [NOMOUSE]
[COLOR <список цветовых пар>/COLOR SCHEME <вырN3>]

Команда активирует все выданные ранее команды @...GET/EDIT и позволяет произвольно перемещаться среди редактируемых полей, используя клавиши Tab/Shift-Tab и клавиши со стрелками. Редактирование может быть завершено (по умолчанию) при попытке продвинуть курсор за первое поле или ниже последнего, а также с помощью клавиш PgUp/PgDn. Выход может быть осуществлен при нажатии клавиш Escape, Ctrl-End/W. В GET-полях доступны все средства редактирования данных, включая удаление, копирование и вставку.

При перемещениях в окнах редактирования осуществляется следующая последовательность действий.

Когда команда READ выполняется первый раз:

исполняется READ WHEN;
активируется первое окно GET;
выполняется READ ACTIVATE;
выполняется READ SHOW;
исполняется опция WHEN уровня GET первой команды @...GET.

Когда активируется новое окно:

выполняется опция VALID для поля, откуда происходит выход;
выполняется READ DEACTIVATE;
деактивируется текущее окно;
активируется новое окно с GET-полями;
выполняется READ ACTIVATE;
выполняется опция WHEN для нового поля.

Опции команды:

CYCLE — запрещает выход из команды READ по достижении первого/последнего объекта, обслуживаемого данной командой внутри окна/экрана. При достижении такого поля курсор перемещается по кругу на последнее/первое поле. Выход из READ остается возможным в случае нажатия клавиш Escape, Ctrl-W/End и использования команды CLEAR READ.

ACTIVATE <вырL1> — организует проверку условий, когда делается попытка войти в очередное окно, управляемое данной командой READ. Результат, возвращаемый <вырL1>, безразличен и не препятствует входу в окно. Условия обычно реализуются в пользовательских процедурах/функциях, которые могут содержать, конечно, не только условия, но и любые действия по управлению (в том числе поиск, предъявление меню, открытие/закрытие других окон, выдача сообщений и т.д.).

DEACTIVATE <вырL2> — то же, но условие проверяется при выходе из текущего окна. Если возвращается значение .T., команда READ завершается, если .F. — осуществляется переход в другое окно.

MODAL — ограничивает доступ пользователя только к окнам, обслуживаемым данной командой **READ**. Другие окна, даже если они открыты, не будут доступны.

WITH <список окон> — задает перечень окон, к которым разрешен доступ, например к окнам **BROWSE**. Опция **MODAL** подразумевается тогда по умолчанию.

SHOW <вырL3> — выполняется всегда при появлении команды **SHOW GETS** и служит обычно для задания процедуры обновления данных, выводимых по командам **@...SAY...GET**, или блокировки/разблокировки **GET**-объектов. Значение, возвращаемое <вырL3>, игнорируется.

VALID <вырL4>/<вырN1> — управляет прекращением команды **READ** и выполняется при попытке выхода из **READ**. Если <вырL4>=.Т., команда завершается, если равно .F. — курсор остается на прежнем или доступном месте. Если опция **VALID** возвращает числовое значение, курсор перемещается на **GET**-объект номер <вырN1>.

WHEN <вырL5> — определяет возможность (<вырL5>=.Т.) входа в команду **READ**. Если <вырL5>=.F., команда **READ** игнорируется, и выполнение программы продолжается со следующей команды.

OBJECT <вырN2> — задает номер **GET**-объекта, на котором будет находиться курсор при входе в команду **READ**. Номер **GET**-объекта определяется последовательностью соответствующих команд **@...GET**.

NOMOUSE — подавляет возможность перемещения среди **GET**-объектов с помощью мыши, хотя не запрещает ее использовать внутри полей.

COLOR <список цветовых пар>/**COLOR SCHEME** <вырN3> — опции устанавливают цвет поля, на котором в данный момент находится курсор. Такое поле получает цвет второй цветовой пары из списка цветовых пар или цветового набора. Остальные цвета игнорируются. По умолчанию используется цветовая схема 1.

Допускается вложение команд **READ** на глубину до пяти уровней. Такое вложение, когда из одной команды **READ** вызывается другая и т.д., можно осуществить с помощью ПФ, вызываемых из опций **VALID**, **WHEN** команды. Имеется функция

■ RDLEVEL()

возвращающая число — текущий уровень вложения команд **READ**. Если возвращен 0, значит, нет активных команд **READ**.

Команда **READ** дает возможность быстрого, произвольного и контролируемого доступа сразу ко многим окнам редактирования.

Разработчики **FoxPro** рекомендуют охватывать одной командой **READ** все окна текущего экрана. Для этого предварительно определенные окна последовательно открываются для размещаемых в них **GET**-объектов по схеме, приведенной ниже. Доступ к этим окнам осуществляется в естественном порядке простым перемещением курсора от объекта к объекту или с помощью клавиш **Ctrl-F1**.

```
ACTIVATE WINDOW <окно1>
@ ... GET ...
ACTIVATE WINDOW <окно2>
@ ... GET ...
READ <опции>
```

Легко можно интегрировать в одном интерфейсе и окна **BROWSE**. Для этого нужно в команду **BROWSE** включить опцию **NOWAIT**, а в **READ** — опцию **WITH** с указанием имени **BROWSE**-окна.

Наделение команд **READ** возможностями управления исключает теперь необходимость в организации программных циклов (обычно циклов **DO WHILE...ENDDO**) по регенерации **GET**-объектов после вызова процедур или

исполнения каких-либо команд (например, для перемещения в базе). Этому способствует наличие опции CYCLE, позволяющей исключить случайное покидание пользователем редактируемого окна, а также специальных команд обновления видимых данных (полей, переменных, органов управления).

По следующей команде осуществляется повторный вывод на экран или в указанное <окно> всех имеющихся на нем GET-объектов:

```
■ SHOW GETS [ENABLE/DISABLE] [LEVEL <вырN1>]
  [OFF/ONLY] [WINDOW <окно>]
  [COLOR <список цветовых пар>/COLOR SCHEME <вырN3>]
```

Опции команды:

ENABLE/DISABLE — разрешает/запрещает дальнейший доступ к объектам и их обновление. В последнем случае они будут показаны "запрещающим" цветом. Опция DISABLE действует только на указанное <окно>. По умолчанию действует ENABLE.

LEVEL <вырN1> — указывает уровень вложенности окна обновляемой команды READ. По умолчанию обновляются GET-объекты текущего уровня READ.

OFF/ONLY — опции предопределяют выполнение (OFF) только процедуры обновления команды READ SHOW <процедура> либо (ONLY) только непосредственное обновление всех GET-объектов, а <процедура> игнорируется. По умолчанию делается и то и другое.

WINDOW <окно> — обновляются GET-объекты только в указанном окне, в противном случае — во всех окнах данной команды READ.

Повторно объекты могут высвечиваться цветом, который устанавливается списком цветовых пар или цветовой схемой (опции COLOR... /COLOR SCHEME ...).

Все GET-объекты получают порядковые номера исходя из последовательности их задания в программе, а не предъявления на экране. Отдельными объектами считаются не только поля/переменные, но и каждая кнопка и другие средства управления из набора, имеющие отдельное представление на экране, пусть и организованные одной командой @...GET, т.е. элементы вида Check Boxes, Invisible Buttons, Push Buttons, Radio Buttons.

Отдельные объекты могут регенерироваться с помощью команд SHOW OBJECT и SHOW GET. Эти команды не связаны с опцией SHOW команды READ. Основное их отличие — это способ указания обновляемого объекта. В первом случае — по его номеру, во втором — по имени, и, возможно, внутреннему номеру.

```
■ SHOW OBJECT <вырN1> [ENABLE/DISABLE]
  [LEVEL <вырN2>] [PROMPT <вырC>]
  [COLOR <список цветовых пар>/COLOR SCHEME <вырN3>]
```

Дополнительные опции:

<вырN1> — абсолютный номер обновляемого объекта, который определяется порядком его описания на экране/окне.

PROMPT <вырC> — задает новую строку-приглашение, заменяющую сделанную в GET-команде.

```
■ SHOW GET <переменная> [, <вырN1> [ENABLE/DISABLE]
  [LEVEL <expN2>] [PROMPT <expC>]]
  [COLOR <список цветовых пар>/COLOR SCHEME <вырN3>]
```

Здесь <переменная> — имя обновляемой переменной/поля GET-объекта. Если для данной <переменной> определено несколько объектов, можно указать относительный номер <вырN1> обновляемого объекта внутри данной

GET-команды. Это гораздо удобнее, поскольку дает возможность программисту не высчитывать номер нужного объекта на экране, а в случае изменения его положения не заботиться об изменениях в программе.

Следующая функция возвращает абсолютный номер GET-объекта по заданному имени <переменной> из текущего или указанного <вырN> уровня READ. Поскольку одной переменной может соответствовать несколько объектов, функция выдает номер самого первого объекта, определяемого данной командой @...GET:

■ OBJNUM(<переменная> [, <вырN>])

Для установления номера текущего объекта можно использовать системную переменную

■ _CUROBJ

значение которой изменяется при переводе курсора на другой объект. В свою очередь, присвоение переменной нового числового значения (_CUROBJ=<вырN>) влечет перемещение курсора на объект номер <вырN>.

Завершить выполнение READ можно по команде вида

■ CLEAR READ [ALL]

которая прекращает команду READ текущего уровня и передает управление на предшествующий уровень READ (если есть). Включение опции ALL вызывает завершение всех команд READ на всех уровнях.

Следующий пример иллюстрирует возможности управления кнопками. Пусть нужно построить кнопочное меню из трех элементов для просмотра в базе KADR.DBF данных отдельно для всех мужчин и женщин следующего вида:

<Вниз> <Вверх> <Женщины/Мужчины>

Кнопки <Вниз> и <Вверх> позволяют перемещаться в базе в указанных направлениях. При этом по достижении конца/начала файла соответствующие кнопки должны быть деактивированы (DISABLE). Третья кнопка должна иметь вид <Женщины>, если просматриваются мужчины, и наоборот, т.е. эта кнопка предоставляет единственно возможную в каждом случае альтернативу выбора фильтра.

Программа приведена ниже. Здесь функция F1() обрабатывает выборы пользователя, т.е. осуществляет перемещения в базе и переустановку фильтра отбора. Вначале предъявляются мужчины (SET FILTER TO rol='M'), а приглашение в кнопке содержит слово "Женщины". При выборе этой кнопки (объект X,1) посредством функции FILTER() выясняется вид действующего фильтра, и он изменяется на противоположный, как и само приглашение. Для запоминания факта достижения конца/начала файла вводятся две переменные E и V. Первоначально они имеют значения .F.. Если была предпринята попытка продвинуться за пределы базы, одна из них получает значение .T.. В остальных случаях они равны .F.. Это нужно для восстановления в процедуре F2() активности кнопок <Вниз>, <Вверх>, если указатель записей находится не на последней/первой записи.

```
SET TALK OFF
USE kadr
STORE .F. TO e,b      && Инициализация переменных конца/начала БД
SET FILTER TO rol='M'  && Устанавливается фильтр
@ 2,2 SAY 'Фамилия' GET fam
@ 4,2 GET x FUNCTION '*NH Вниз;Вверх;Женщины' VALID f1() DEFAULT 1
READ CYCLE SHOW f2()
```

```
FUNCTION f2      &&-----Функция восстановления кнопок
IF !e           && Если указатель не на последней записи,
  SHOW GET x,1 ENABLE  && активируется кнопка <Вниз>
```



```

ENDIF
IF !b                                && Если указатель не на первой записи,
    SHOW GET x,2 ENABLE              && активируется кнопка <Вниз>
ENDIF
RETURN

FUNCTION f1                          &&-----Функция, обработки кнопок
DO CASE
CASE x=1                              && <Вниз>
    SKIP
    IF EOF()                          && Если конец файла,
        SHOW GET x,1 DISABLE          && кнопка деактивируется,
        GO BOTTOM                      && возврат на конечную запись и
        e=.t.                         && запоминание этого факта
    ELSE                              && Если нет,
        STORE .f. TO e,b              && переменные очищаются
    ENDIF
CASE x=2                              && <Вверх>
    SKIP -1
    IF BOF()                          && Если начало файла,
        SHOW GET x,2 DISABLE          && кнопка деактивируется,
        GO TOP                        && возврат на начальную запись
        b=.t.                         && и запоминание этого факта
    ELSE                              && Если нет,
        STORE .f. TO e,b              && переменные очищаются
    ENDIF
CASE x=3                              && <Мужчины/Женщины>
    IF FILTER()=[POL="M"]              && Если отображены Мужчины,
        SET FILTER TO pol='Ж'         && фильтр переключается и
        SHOW GET x,3 PROMPT 'Мужчины' && кнопка заменяется
    ELSE                              && Если нет, обратная операция
        SET FILTER TO pol='М'
        SHOW GET x,3 PROMPT 'Женщины'
    ENDIF
    GO TOP
    STORE .f. TO e,b                  && Переменные очищаются
ENDCASE
SHOW GETS                             && Обновление поля FAM и кнопок
RETURN

```

Теперь рассмотрим пример совмещения в одном окне нескольких GET-объектов. Ранее в гл.22 была рассмотрена программа, где для базы KADR.DBF с помощью макроподстановки формировался гибкий ключ поиска данных. Однако возможности пользователя по образованию такого критерия были весьма малы. Он мог только последовательно его сужать, соединяя компоненты критерия логическим оператором И (.AND.). Хотя это и самый распространенный случай, но может понадобиться любой способ доступа, включающий все логические и арифметические операции и операторы отношения. Ниже приведена программа формирования ключа поиска, которая порождает рабочее окно, изображенное на рис.26.1.

(Дата рожд. <= '01.01.28' ИЛИ Пол = 'Ж' И Дата рожд. <= '01.01.33') И Подразделение = 'ОГМ'

Операторы: ИЛИ Поля: Фамилия

<Поиск>
<Отказ>
<Очистка>

(dtr<='{01.01.28}'.OR.pol='Ж'.AND.dtr<='{01.01.33'}).AND.
 .podr='ОГМ'

-----Строки и даты вводятся в апострофах-----

Рис.26.1

Окно состоит из четырех основных элементов.

1. В верхней части отображается собственно поле, в котором формируется критерий в переменной U. Допускается как непосредственный ввод данных "руками", так и занесение их через доступные меню.

2. POPUP-меню "Операторы" содержит перечень всех знаков операций,

которые могут понадобиться пользователю (ИЛИ, И, НЕ, +, -, = и т.д.).

3. POPUP-меню "Поля" включает содержательные русские названия всех полей (кроме мемо-поля).

4. Средства управления в виде кнопок.

Кнопка <Поиск> формирует из введенного пользователем словесного критерия его формальный аналог на языке FoxPro, осуществляет проверку критерия на синтаксическую корректность и выполняет сам поиск нужных записей. В поле критерия мы видим уже некоторую сформированную строку запроса. В данном случае нужно найти пенсионеров по старости на 1 января 1993 г., работающих в отделе главного механика (ОГМ). На основе такого содержательного критерия формируется формальный критерий, который и отражается в нижней части окна.

Кнопка <Отказ> завершает работу с окном без последствий.

Кнопка <Очистка> вызывает освобождение переменной U от каких-либо данных, например если задается совершенно новый критерий или если выяснилось, что критерий введен с ошибками и пользователю кажется более удобным ввести его заново, а не заниматься правкой.

Сама программа приведена ниже. Здесь сначала создается массив A(8,2), в который вводятся названия полей, что необходимо для второго меню. В переменную Z вносятся все нужные знаки операций для первого POPUP-меню. Затем определяется и открывается окно KRIT для создания критерия.

Под переменную критерия U отводится три строчки общей длиной 228 символов. Функция UF() не допускает образования пробелов после ввода очередного компонента.

Меню "Операторы:" формирует числовую переменную OP и "обслуживается" функцией OPF(), где сначала выясняется будущая длина U (под знак операции отводится три позиции). Если она не превышает отводимых под переменную U двухсот двадцати восьми позиций, знак присоединяется к U. В противном случае выводится предупреждение и дополнение прекращается. После обновления поля U (SHOW GET u) по командам _CUROBJ=1 и KEYBOARD '{end}' курсор передвигается в поле U (объект номер 1) в конец строки. Это сделано для экономии времени пользователя, поскольку обычно после указания знака операции осуществляется непосредственный ввод некоторой константы. Таким образом отпадает необходимость в перемещении маркера мыши.

Меню "Поля" формирует переменную PL, которая обрабатывается в функции PLF(). Функция в общем аналогична OPF(), но относится к названиям полей. Поскольку длина названия не постоянна, оно сначала заносится в переменную S и только после этого определяется будущий размер U.

Кнопки управления определяют переменную C и обрабатываются в функции CF(). Среди них кнопка <Поиск> назначается кнопкой по умолчанию. Такая кнопка может быть выбрана из любого положения курсора нажатием клавиш Ctrl-Enter. Кнопка <Отказ> действует аналогично, но в результате нажатия клавиши Escape. Функции кнопок <Отказ> и <Очистка> очевидны (с=2 и с=3). Кнопка <Поиск> (с=1) из переменной U формирует формальный критерий поиска в переменной UU. Содержательные названия полей заменяются их именами, знаки операций И, ИЛИ, НЕ — на строки .AND., .OR., .NOT.. Дата, введенная в апострофах, берется в фигурные скобки. Для этого в строке критерия разыскивается подстрока длиной в восемь символов, взятая в апострофы и разделенная точками через каждые два разряда. Конечно, можно было бы обязать пользователя использовать в таком случае именно скобки, однако надеяться на это не стоит. Хорошо, если он не забудет и про апострофы.

Важнейшей командой здесь является IF !TYPE("&uu")='L', с помощью

которой осуществляется проверка синтаксической правильности построения критерия. Если полученный тип выражения логический ('L'), значит, все в порядке, если нет значит, при вводе критерия была допущена ошибка и пользователь возвратится в окно. Здесь обратите внимание на то, что макроподстановка взята в кавычки, а не в апострофы. Это вызвано тем, что апострофы уже есть внутри выражения UU.

Далее при правильно заданном критерии ищется первая подходящая запись (LOCATE FOR &uu). Если такая запись находится, окно временно удаляется и на экран выводятся все нужные записи с последующим восстановлением окна.

*-----Программа ввода критерия поиска для базы KADR.DBF-----

```
CLEAR
SET TALK OFF
SET DATE GERMAN
DIMENSION f(8,2)      && Массив содержательных названий (столбец 1)
f(1,1)='Фамилия'      && и имен полей базы KADR.DBF (столбец 2)
f(2,1)='Табель'
f(3,1)='Дата рожд.'
f(4,1)='Пол'
f(5,1)='Сем. положение'
f(6,1)='К-во детей'
f(7,1)='Подразделение'
f(8,1)='Ср. зарплата'
f(1,2)='fam'
f(2,2)='tab'
f(3,2)='dtr'
f(4,2)='pol'
f(5,2)='sem'
f(6,2)='det'
f(7,2)='podr'
f(8,2)='szar'
* Возможные виды операторов:
z='ИЛИ;И ;НЕ ;+ ;- ;* ;/ ;> ;>= ;< ;<= ;# ;= ;( ;)'
DEFINE WINDOW krit FROM 3,1 TO 13,79;
FOOTER 'Строки и даты вводятся в апострофах'
ACTIVATE WINDOW krit
USE kadr
u='      && Переменная, где формируется содержательный ключ поиска
@ 0,1 GET u SIZE 3,76 VALID uf()      && Поле ввода переменной U
@ 4,6 SAY 'Операторы:'      && Меню операций
@ 3,17 GET op FUNCTION '~ ' +z VALID opf() DEFAULT 1
@ 4,38 SAY 'Поля:'      && Меню полей
@ 3,45 GET pl FUNCTION '~ ' FROM f DEFAULT 1 VALID plf()
@ 6,15 GET c FUNCTION '~HN \!Поиск;\?Отказ;Очистка' VALID cf();
DEFAULT 1 SIZE 1,5,10      && Кнопки завершения
READ CYCLE
```

```
FUNCTION uf      &&----- Функция вывода значения U
u=RTRIM(u)      && Удаление завершающих пробелов
SHOW GET u      && Обновление видимого поля U
RETURN
```

```
FUNCTION opf      &&----- Функция дополнения U знаком операции
* Если общая длина U больше отведенного для нее в окне
* места (228 символов), дополнение прекращается
IF LEN(u)+3>228
WAIT 'Критерий слишком длинный' WINDOW NOWAIT
RETURN
ENDIF
u=u+' '+RTRIM(SUBSTR(z,(op-1)*4+1,3))
```

```
SHOW GET u
CURSOR=1
KEYBOARD '{end}'      && Установление курсора в конец строки для
RETURN      && ручного ввода параметра
```

```
FUNCTION plf      &&----- Функция дополнения U именем поля
s=' '+f(pl,1)
IF LEN(u)+LEN(s)>=228      && Проверка полученной длины U
WAIT 'Критерий слишком длинный' WINDOW NOWAIT
RETURN
```



```

ENDIF
u=u+s
SHOW GET u
RETURN

FUNCTION cf      &&----- Функция завершения формирования U
DO CASE
CASE c=2      && Если выбрана кнопка <Отказ>,
CLEAR READ      && сеанс завершается без последствий
RELEASE WINDOW krit
CASE c=3      && Если выбрана кнопка <Очистка>,
u=' '          && сбрасывается переменная U для
SHOW GET u      && повторного ввода
@ 7,0 CLEAR
CASE c=1      && Если выбрана кнопка <Поиск>,
@ 7,0 CLEAR
uu=u          && создается переменная UU для критерия
FOR i=1 TO 8    && Если выбрана кнопка <Поиск>,
uu=STRTRAN(uu,f(i,1),f(i,2)) && содержательные названия
ENDIF          && замещаются на имена полей
uu=STRTRAN(uu,'ИЛИ','OR.') && Русские названия логических
uu=STRTRAN(uu,'И','AND.') && операций замещаются на
uu=STRTRAN(uu,'НЕ','NOT.') && принятые в FoxPro
uu=STRTRAN(uu,' ') && Удаляются ненужные пробелы
FOR i=1 TO 50   && Поиск даты
n1= AT([''],uu, i) && Позиция очередного апострофа
n2= AT([''],uu, i+1) && Позиция следующего апострофа
IF n2=0         && Если больше апострофов нет,
EXIT           && завершение обработки
ENDIF
IF n2-n1=9.AND.SUBSTR(uu,n1+3,1)=;
'. ' .AND.SUBSTR(uu,n1+6,1)='.'
* Если найдены апострофы, отстоящие друг от
* друга на 9 символов, где в третьей и шестой
* позициях содержатся точки, значит, это дата
uu=STUFF(uu,n1,1,'{') && Тогда апострофы заменяются на
uu=STUFF(uu,n2,1,'}') && фигурные скобки
i=i+1          && Увеличение номера апострофа
ENDIF
ENDFOR
@ 7,0 SAY LEFT(uu,76) && Вывод полученного критерия
@ 8,0 SAY SUBSTR(uu,77,76) && на языке FoxPro
IF !TYPE("&uu")='L' && Проверка его синтаксической правильности
WAIT 'Критерий поиска задан ошибочно' WINDOW NOWAIT
RETURN
ELSE
LOCATE FOR &uu && Если UU задано корректно,
&& выполняется поиск
IF !FOUND() && Если такой записи нет, сообщение
WAIT 'Поиск неудачный' WINDOW NOWAIT
ELSE && Иначе -
HIDE WINDOW krit && Временное скрытие окна
ACTIVATE SCREEN && Активируется экран и на него
CLEAR
DISPLAY REST fam,tab,dtr,pol,sem,podr;
FOR &UU && выводятся найденные записи
WAIT
SHOW WINDOW krit && Восстановление окна
ENDIF
ENDIF
ENDCASE
RETURN
*-----Конец программы-----

```

Следующая программа дает некоторое представление о создании многооконного интерфейса в стиле Windows. Задача заключается в том, чтобы отобразить на одном экране окно, содержащее сведения о личности работника из базы KADR.DBF (окно KD), окно (BR) со всеми выработками данного человека во всех бригадах (из файлов вида BRIG*.DBF) и окно (DG), содержащее диаграммы выработок работника. Этот экран приведен ниже.

В окне KD кроме собственно данных представлены и средства управления. В правом углу отображено скрытое POPUP-меню, содержащее наименования

всех подразделений предприятия. Обратившись к нему, можно выбрать и сразу же занести подразделение в поле PODR базы KADR.DBF. В центре окна отображены кнопки управления перемещением в базе и выхода из нее. Кроме того, кнопка <Итого> позволяет вывести на экран полную зарплату работника с учетом различных надбавок. Установка этих надбавок выполняется в разделе окна "Доплаты:". Здесь могут быть указаны доплаты за вредные условия работы, а также доплаты инвалидам и участникам войны.

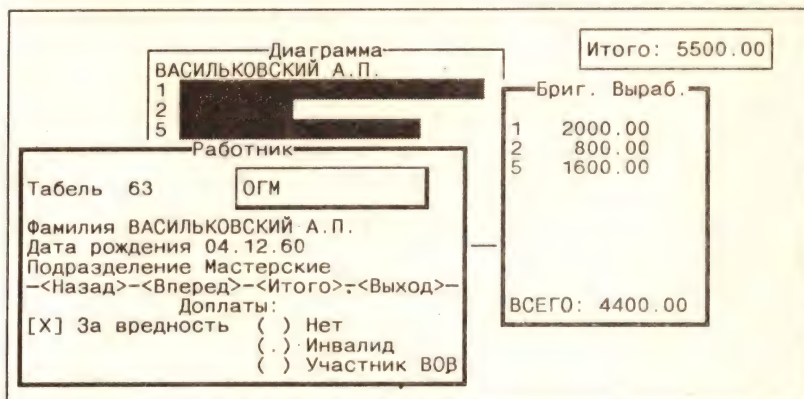


Рис.26.2

В окне BR отображены поля выработок (VIR) для всех найденных на диске бригадных файлов, а также номера бригад. Если работник, данные о котором отображены в левом окне, что-то заработал в составе этих бригад, здесь будут отображены соответствующие цифры, если нет — нули. С этой целью все бригадные файлы сцеплены с базой KADR.DBF по полю TAB. Чтобы поддержать связь, файлы BRIG*.DBF проиндексированы по полю TAB (создаются одноименные индексные файлы), но, конечно, с расширением IDX, а не DBF. Кроме того, в нижней части экрана показывается суммарная выработка работника по всем бригадам. Программа, реализующая этот интерфейс, приведена ниже.

В окне DG отображается практически та же информация, что и в окне BR, но в виде горизонтальных столбиковых диаграмм.

Базы KADR.DBF и PODR.DBF открываются в областях 1 и 2. Далее объявляется массив PDR(30,1) для перенесения в него всех непустых полей PODR из базы PODR.DBF. Поскольку подразделений может быть и меньше 30, после копирования массив снова переопределяется (без потери данных) по команде DIMENSION, но уже для фактического числа найденных подразделений (_TALLY). Это позволит нам избавиться от бесполезных затрат памяти на хранение пустых элементов массива, а главное, не допустить включения таких элементов в меню.

Далее инициализируются переменные под сумму (S) и признак доплат (DP), а также вводится переменная (BMAX), в которую в дальнейшем будет заноситься максимальная выработка данного работника в бригаде. Эта переменная нужна для масштабирования диаграмм в окне DG. В переменную L заносится количество обнаруженных на диске файлов вида BRIG*.DBF, а в массив В — их имена и другие реквизиты. Имена таких файлов нам понадобятся для их открытия в последовательных рабочих областях, начиная с третьей, совместно с индексами. Это действие выполняется в цикле (FOR i=1 TO L). Здесь же устанавливаются и реляционные связи. Для удобства

пользователя номера бригад будут выводиться в порядке возрастания, что обеспечивается функцией ASORT(). Подготовительные действия завершаются определением окон KD, BR и DG, которые можно перемещать (FLOAT).

Далее формируются все GET-объекты интерфейса последовательно в окнах KD и BR. Собственно окна активируются первоначально как скрытые (NOSHOW), и в них выводятся данные. Их предъявление осуществляется только после их заполнения всеми данными и элементами интерфейса по команде SHOW WINDOW непосредственно перед командой READ. Если этого не сделать, пользователь станет свидетелем формирования экрана, когда поля будут предъявляться последовательно одно за другим, а так он увидит сразу готовые окна.

GET-переменная управления K обрабатывается в функции KK(), где осуществляются перемещение в базе, выход из нее и вычисление суммы с доплатами. В последнем случае определяющими являются переменные DP и VR. В зависимости от их значений начисляется прибавка к зарплате 15, 10 и 10 % для инвалидов, участников войны и за вредные условия работы. Эти величины добавляются к суммарной выработке работника и предъявляются на экране по команде WAIT.

Скрытое POPUP-меню формируется из массива PDR, а выбор из меню обрабатывается в функции PD(), где нужно подразделение (соответствующий элемент массива) переносится в поле PODR.

Сама команда READ "нагружена" вычислением суммарного значения выработок сотрудника в бригадах и выводом диаграмм. Для этого в опции SHOW вызывается функция SUM(), где происходит сложение выработок в переменной S и выдача ее в последнюю строку окна BR. Попутно вычисляется максимальное значение выработки в бригаде (BMAX), на основании которой определяется масштабный коэффициент K. Сам график выводится в цикле FOR I+1 TO L. По завершении выдачи диаграмм осуществляется возврат в окно, из которого произошел вызов процедуры.

Поскольку данные о выработках могут пользователем изменяться, необходимо сделать так, чтобы при выходе окна BR (т.е. при WLAST()='BR') сумма пересчитывалась и обновлялась снова. Это действие реализовано через опцию DEACTIVATE команды READ. Чтобы не дать завершиться команде READ по окончании опции, она "заперта" логическим значением .F.. Этот же результат мог быть получен замыканием процедуры SUM() командой RETURN .F., однако сделать этого нельзя, поскольку процедура используется и в другом месте, где нужно вырабатывать значение .T. (действует по умолчанию).

```
*----- Программа KADR5.PRG Нужны файлы: KADR.DBF, PODR.DBF-----
*----- и бригадные файлы BRIG*.DBF и BRIG*.IDX-----
SET TALK OFF
SET DATE GERMAN
CLEAR
USE kadr IN 1      && Открытие базы KADR.DBF
USE pdr IN 2       && Открытие базы PODR.DBF
DIMENSION pdr(20,1) && Создание массива подразделений для меню
SELECT 2           && Копирование подразделений в массив
COPY TO ARRAY pdr FIELDS b.pdr FOR !EMPTY(b.pdr)
SELECT 1
DIMENSION pdr(_TALLY,1) && Уточнение размеров массива
s=0                && Полная заработанная сумма
dp=0               && Доплаты
bmax=0             && Максимальная выработка в бригаде
l=ADIR(b,'brig*.dbf') && Заполнение массива В реквизитами
                     && баз бригад BRIG*.DBF
=ASORT(b,1)        && Сортировка массива по номерам бригад
FOR i=1 TO l       && Открытие этих баз и установление связей
  USE (b(i,1)) IN i+2 INDEX (LEFT(b(i,1),5))
  SET RELATION TO a.tab INTO i+2 ADDITIVE
```



```

ENDFOR
DEFINE WINDOW kd FROM 7,0 TO 19,35 TITLE 'Работник' DOUBL FLOAT
DEFINE WINDOW br FROM 4,40 TO 16,57;
TITLE 'Бриг. Выrab.' DOUBL FLOAT
DEFINE WINDOW dg FROM 2,11 TO 12,44 TITLE 'Диаграмма' FLOAT COLOR n/w
*-----Формирование интерфейса-----*
ACTIVATE WINDOW kd NOSHOWN && Окно KD (данные о работнике)
@ 1,0 SAY 'Табель' GET tab
@ 3,0 SAY 'Фамилия' GET fam
@ 4,0 SAY 'Дата рождения' GET dtr
@ 5,0 SAY 'Подразделение' GET podr
@ 6,0 TO 6,34
@ 6,1 GET k PICTURE "@*HN Назад;Вперед;Итого;Выход";
DEFAULT 1 VALID kk()
@ 0,18 GET p FUNCTION '' FROM pdr VALID pd() DEFAULT 1
@ 7,13 SAY 'Доплаты:'
@ 8,0 GET vr FUNCTION '*С За вредность' DEFAULT .f.
@ 8,18 GET dp PICTURE '@*RV Нет;Инвалид;Участник БОВ' DEFAULT 1
ACTIVATE WINDOW br NOSHOWN && Окно BR (данные по бригадам)
FOR i=1 TO l && Вывод полей VIR для сцепленных
bb=SUBSTR(b(i,1),5,1) && по полю TAB бригадных файлов
bbb='brig'+bb+'.vir'
@ i,0 SAY bb+' ' GET (bbb)
ENDFOR
ACTIVATE WINDOW dg NOSHOWN && Окно DG (диаграмма)
SHOW WINDOW br,dg,kd && Предъявление окон
READ CYCLE SHOW sum();
DEACTIVATE IIF(WLAST()='BR',sum(),.f.).AND..f.
DEACTIVATE WINDOW br,kd,dg

FUNCTION pd &&--Функция выбора названия подразделения из меню
REPLACE podr WITH pdr(p,1)
SHOW GET a.podr && Обновление поля A.PODR
RETURN

FUNCTION sum &&--Функция суммирования всех выработок работника
w=WONTOP() && Запоминание имени текущего окна
s=0
bmax=0
FOR i=1 TO l
v=EVALUATE('brig'+SUBSTR(b(i,1),5,1)+' .vir') && Выработка
s=s+v && Вычисление суммарной выработки
b(i,2)=v && Запоминание выработки в массиве
IF bmax<b(i,2) && Нахождение макс. выработки для масштаба
bmax=b(i,2)
ENDIF
ENDFOR
ACTIVATE WINDOW br && Вывод суммы в окно BR
@ 10,0 SAY 'BCEFO: '+STR(s,8,2)
ACTIVATE WINDOW dg SAME && Активируется окно диаграмм DG
@ 0,0 SAY a.fam
IF bmax=0 && Если выработка равна нулю,
@ 1,0 CLEAR && окно диаграмм очищается
ELSE && Если нет, вычисляется
k=24/bmax && масштабный коэффициент
FOR i=1 TO l && и выводятся диаграммы
@ $+1,0
@ $,0 SAY SUBSTR(b(i,1),5,1)+:
'+REPLICATE('█',ROUND(b(i,2)*k,2))
ENDFOR
ENDIF
ACTIVATE WINDOW (w) && Возврат в текущее окно
RETURN,

FUNCTION kk &&-----Функция обработки кнопок управления
DO CASE
CASE k=2 && <Вперед>
SKIP
IF EOF()
GO BOTTOM
ENDIF
SHOW GETS WINDOW br && Обновление окон
SHOW GETS WINDOW kd
CASE k=1 && <Назад>
SKIP -1

```



```

        IF BOF()
            GO TOP
        ENDIF
        SHOW GETS WINDOW br          && Обновление окон
        SHOW GETS WINDOW kd
CASE k=4                                && <Выход>
    CLEAR READ
CASE k=3                                && <Итого> - расчет полного заработка
    kk=0
    DO CASE                            && Доплаты инвалидам и участникам войны
        CASE dp=2
            kk=0.15
        CASE dp=3
            kk=0.10
    ENDCASE
    IF vr                                && Доплата за вредные условия работы
        kk=kk+0.1
    ENDIF
    WAIT 'Итого: ' +STR(s+s*kk,8,2) WINDOW && Вывод результата
ENDCASE
RETURN
*-----Конец модуля-----

```

Рассмотрим пример ведения базы данных по учету занятости мест в гостинице. База GOST.DBF состоит из следующих семи полей:

- занимаемое место в гостинице (MES);
- фамилия проживающего (FAM);
- пол (POL);
- дата прибытия в гостиницу (PRI);
- предполагаемая дата отбытия (OTB);
- дата, по которую оплачено проживание в номере (OPL).

Номер занимаемого места состоит из номера этажа (одна цифра), номера комнаты (две цифры) и собственно номера места в комнате.

В соответствии с этим база имеет такую структуру:

Field	Field Name	Type	Width
1	MES	Character	4
2	FAM	Character	25
3	POL	Character	1
4	PRI	Date	8
5	OTB	Date	8
6	OPL	Date	8

База проиндексирована (индекс GOST.IDX) по полю MES:

```
INDEX ON mes TO gost COMPACT
```

Особенностью предлагаемой системы является доступ не только к собственно базе данных, но и к поэтажному плану гостиницы с возможностью здесь же получить данные о проживающем.

Интерфейс системы с конкретным наполнением приведен на рис.26.3.

Он состоит из BROWSE-окна, где предъявляются все данные о клиентах, и нескольких окон (по числу этажей) с планом каждого этажа. В данном примере рассмотрены два этажа.

Программа GOST.PRG приведена ниже. BROWSE-окно имеет два вычисляемых поля. В поле D (Долг) выводится предупреждение "Долг", если текущая дата больше даты, по которую оплачен номер. В поле R (вторая колонка таблицы) отображается курсор-дубликат в виде знака ">". Необходимость в нем существует ввиду того, что, когда активны окна этажей (E1 или E2), из-за отсутствия курсора в окне GOST невозможно определить, кто занимает рассматриваемое на плане гостиницы место. Для поля POL делается не только

обычная проверка на допустимое значение "М" или "Ж", но и выполняется просмотр данных о всех проживающих жильцах этой комнаты (функция PL()). Если выясняется, что вновь заселяемый мужчина, а в комнате проживает женщина, выводится предупреждение 'В комнате живет Женщина', и наоборот. Выдается именно предупреждение, а не запрет, поскольку в комнату могут быть поселены и супруги. Очевидно, что такая проверка облегчает администратору гостиницы размещение прибывающих постояльцев.

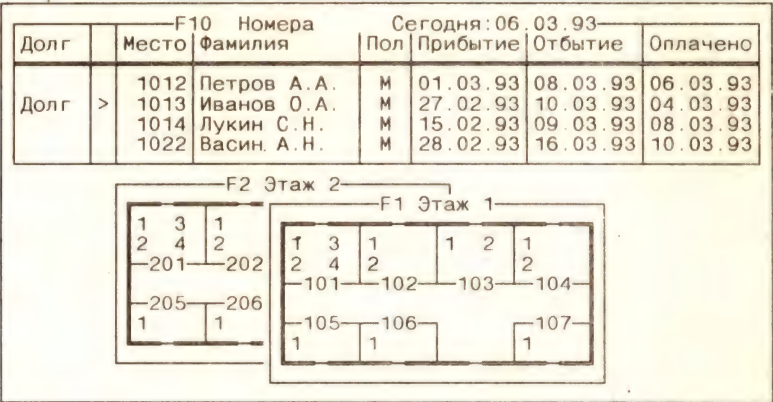


Рис.26.3

В нижней половине экрана изображены планы этажей. Каждое место в комнате имеет свой номер, по которому можно выяснить данные о занимающем его жильце. С этой целью все номера "накрыты" невидимыми экранными кнопками (Invizible Buttons), которые образуют меню из этих номеров. Все кнопки "обслуживаются" одной VALID-функцией MS(), куда в качестве параметра передается символьный номер места. Далее он разыскивается (SEEK) в индексном поле базы данных. В переменной R запоминается номер найденной записи для формирования дополнительного курсора и обновляется BROWSE-окно.

Рассмотренный механизм обеспечивает быстрое отображение в BROWSE-окне сведений о любом жильце, проживающем на месте, которое мы выберем на плане.

```
*-----Программа GOST.PRG-----
*-----Нужны файлы GOST.DBF и GOST.IDX-----
SET TALK OFF
SET DATE GERMAN
ON KEY LABEL F10 ACTIVATE WINDOW F10      && Вызов окна базы
ON KEY LABEL F1 ACTIVATE WINDOW e1        && Вызов окна первого этажа
ON KEY LABEL F2 ACTIVATE WINDOW e2        && Вызов окна второго этажа
USE gost INDEX: gost
r=RECNO()
DEFINE WINDOW gost FROM 0,0 TO 8,62;
  TITLE 'F10 Номера'                      && Окно-список номеров
DEFINE WINDOW e1 FROM 10,16 TO 19,42;
  TITLE 'F1 Этаж 1' COLOR n/gb            && Окно первого этажа
DEFINE WINDOW e2 FROM 9,4 TO 18,30;
  TITLE 'F2 Этаж 2' COLOR n/gb            && Окно второго этажа
BROWSE TITLE 'F10 Номера Сегодня: '+DTOC(
FIELD;
d=IIF(
r=IIF(
mes:h='Место':p:'9999';
fam:h='Фамилия':14;
pol:h='Пол':v=(pol='М'.OR.pol='Ж').AND.pl(LEFT(mes,3),pol);
```



```

:e='Только М или Ж',;
pri :h='Прибытие',;
otb :h='Отбытие',;
opl :h='Оплачено',;
COLOR SCHEME 10 WINDOW gost NOWAIT WHEN ttt()

```

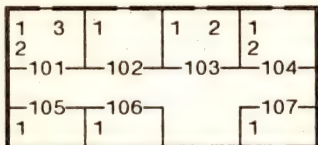
ACTIVATE WINDOW e1

&& Открытие окна первого этажа

```

@ 0,0 SAY
@ 1,0 SAY
@ 2,0 SAY
@ 3,0 SAY
@ 4,0 SAY
@ 5,0 SAY
@ 6,0 SAY
@ 7,0 SAY
@ 1,1 GET n FUNCTION '*I' SIZE 1,1 DEFAULT 1 VALID ms('1011')
@ 2,1 GET n FUNCTION '*I' SIZE 1,1 DEFAULT 1 VALID ms('1012')
@ 1,4 GET n FUNCTION '*I' SIZE 1,1 DEFAULT 1 VALID ms('1013')

```



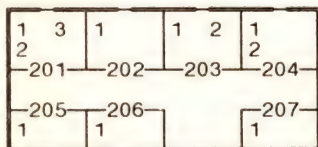
ACTIVATE WINDOW e2

&& Открытие окна второго этажа

```

@ 0,0 SAY
@ 1,0 SAY
@ 2,0 SAY
@ 3,0 SAY
@ 4,0 SAY
@ 5,0 SAY
@ 6,0 SAY
@ 7,0 SAY
@ 1,1 GET n FUNCTION '*I' SIZE 1,1 DEFAULT 1 VALID ms('2011')
@ 2,1 GET n FUNCTION '*I' SIZE 1,1 DEFAULT 1 VALID ms('2012')

```



```

READ CYCLE
DEACTIVATE WINDOW e1,e2,F10
RELEASE WINDOWS e1,e2,F10
ON KEY

```

```

FUNCTION ttt      &&-----Функция обновления курсора
r=RECNO()
SHOW WINDOW F10 REFRESH
RETURN

```

```

FUNCTION ms      &&-----Функция отслеживания в окне GOST
&& записи с информацией о человеке, занимающем место,
&& где находится курсор на плане этажа
PARAMETERS m
SEEK m          && Поиск в BROWSE-окне выбранного на плане места
r=RECNO()
SHOW WINDOW F10 REFRESH
RETURN

```

```

FUNCTION pl      &&-----Функция предупреждений (Мужчина/Женщина)
PARAMETERS m,p
nr=RECNO()      && Запоминается номер текущей записи
&& Поиск непустого места в той же комнате, где проживает
&& человек с другим значением поля POL
LOCATE FOR mes=m.AND.!EMPTY(pol).AND.pol#p
IF FOUND()      && Если поиск удачный, выдается предупреждение
WAIT 'В комнате живет '+IIF(pol='M','Мужчина','Женщина');
WINDOW NOWAIT
GO nr          && Возврат к исходной записи
ENDIF
RETURN

```

*-----Конец модуля-----

Конечно, желательно иметь возможность не только перемещаться по плану гостиницы, но выделять на нем свободные места, места оплаченные "по сегодня", и места должников. Кроме того, следует учитывать, что реальная вместимость гостиницы может быть весьма значительной. Легко представить себе отель, имеющий двадцать этажей и тысячу мест. Очевидно, "нагрузить" команду READ тысячами кнопок вряд ли даже возможно, не говоря уже о том, что программа, где каждой кнопке соответствует отдельная команда, будет

очень громоздкой.

Здесь можно предложить следующее решение. Во-первых, введем еще одну базу PLAN.DBF, содержащую единственное мемо-поле (тоже с именем PLAN), где в каждой записи будут храниться поэтажные планы гостиницы. Во-вторых, добавим в базу GOST.DBF два числовых поля Y и X (размерностью два разряда целых), в которых будут храниться оконные координаты каждой кнопки. Поскольку поддерживать множество больших окон для компьютера довольно обременительно, придется ограничиться единственным окном PLAN, куда будем проектировать план любого этажа. С тем чтобы иметь возможность "путешествовать" по этажам, в левом верхнем углу окна выведена кнопка-меню "Этажи". Программа GOST1.PRG приведена ниже.

```
*-----Программа GOST1.PRG-----
*--Нужны файлы: GOST.DBF, GOST.IDX, PLAN.DBF, PLAN.FPT-----
SET TALK OFF
SET DATE GERMAN
SET ESCAPE OFF
&& Установление цветовых схем для кнопок в окне PLAN
SET COLOR OF SCHEME 15 TO ,,,w+*/g ,,,w+/g    && Место свободно
SET COLOR OF SCHEME 18 TO ,,,w+*/gb ,,,w+/gb   && Место занято
SET COLOR OF SCHEME 16 TO ,,,w+*/b ,,,w+/b     && Оплата сегодня
SET COLOR OF SCHEME 17 TO ,,,w+*/r ,,,w+/r     && Должник
l=1
z=1
mm=30
DIMENSION nn(mm)
nn=''
ON KEY LABEL F10 ACTIVATE WINDOW F10
ON KEY LABEL F2 ACTIVATE WINDOW plan
USE gost IN a INDEX gost
USE plan IN b
r=RECNO()
DEFINE WINDOW gost FROM 0,0 TO 8,62;
    TITLE 'F10 Номера'                                && Окно-список номеров
DEFINE WINDOW plan FROM 9,4 TO 23,31;
    TITLE 'F2 План этажа' COLOR SCHEME 10            && Окно этажа
BROWSE TITLE 'F10 Номера Сегодня: '+DTOC(DATE());
    FIELD;
    d=IIF(DATE()>opl.AND.!EMPTY(fam),'Долг','') :h='Долг',;
    r=IIF(r=RECNO(),'>','') :h='';
    mes :h='Место' :p='9999',;
    fam :h='Фамилия' :14,,;
    pol :h='Пол',;
    pri :h='Прибытие',;
    otb :h='Отбытие',;
    opl :h='Оплачено',;
    COLOR SCHEME 10 WINDOW gost NOWAIT WHEN ttt()
ACTIVATE WINDOW plan                                && Открытие окна этажа
@ 1,0 SAY 'Этаж:'                                && Формирование кнопки-меню
@ 0,6 GET l FUNCTION '~ 1;2;3;4;5;6;7' VALID et(l)
@ 0,11 SAY 'Место свободно' COLOR w+/g
@ 1,11 SAY 'Место занято' COLOR w+/gb
@ 2,11 SAY 'Оплата сегодня' COLOR w+/b
@ 3,11 SAY 'Должники' COLOR w+/r
=et(1)
READ CYCLE
DEACTIVATE WINDOW plan,F10,gost
RELEASE WINDOWS plan,F10,gost
CLEAR READ
ON.KEY

FUNCTION et
PARAMETERS e
@ 4,0 CLEAR
GO e IN b
@ 4,0
??b.plan
* Поиск в базе GOST.DBF первого места на этом этаже
SEEK LTRIM(STR(e))
```



```

nr=RECNO()          && Запоминание номера текущей записи
i=2                 && Начальный элемент заполняемого массива
nn=''
* Сканирование всех мест на этаже
SCAN WHILE a.mes=LTRIM(STR(e))
  @ a.y+4,a.x GET z FUNCTION '*I' SIZE 1,2 DEFAULT 1;
  COLOR SCHEME IIF(EMPTY(a.fam),15,IIF(a.opl<DATE(),17,;
  IIF(a.opl=DATE(),16,18)));
  VALID ms(nn(_Curobj))          && Вывод кнопок
  nn(i)=a.mes                    && Заполнение массива номерами мест
  i=i+1
ENDSCAN
GO nr                    && Возврат на исходную запись
RETURN
*-----Конец модуля-----

```

Поскольку предполагается различная раскраска кнопок в зависимости от того, свободно место или занято, а также имеется ли задолженность у жильца, в начале программы устанавливаются две (значащие для кнопок) цветовые пары (шестая и девятая) четырех цветовых схем (15, 16, 17, 18). Далее объявляется массив NN(MM), где будут храниться номера мест для каждой из кнопок на этаже. Здесь MM=30, т.е. предполагается до 30 мест на этаже. Затем открываются BROWSE-окно и окно PLAN, в котором создается кнопка-меню с номерами этажей (здесь семь) и для пользователя выводятся образцы раскраски кнопок. Непосредственно перед командой READ выполняется функция ET(), формирующая наполнение окна PLAN. Эта же функция используется для обработки выбора этажа. Рассмотрим ее.

В функцию передается числовой параметр (E), соответствующий номеру этажа, план которого желательно отобразить. Это число одновременно является номером записи базы PLAN.DBF (область B), где в мемо-поле PLAN хранится карта этажа. После того как нужная запись найдена, окно PLAN очищается с четвертой строки и на это место выводится план. Выше четвертой строки находится кнопка управления, которая, естественно, не стирается. Далее в базе GOST.DBF ищется первая запись, соответствующая искомому этажу. Начиная с нее, сканируются все записи, для которых первая цифра в поле MES совпадает с номером заданного этажа (E). Внутри цикла в координатах Y+4 и X, считанных из базы GOST.DBF, выводятся невидимые кнопки. Каждая такая кнопка получает цвета из цветовой схемы, определенной в зависимости от того, свободно место (EMPTY(a.fam)), имеется долг у клиента (opl<DATE()), истекает ли оплаченное время (opl=DATE()). Все невидимые кнопки смещены по вертикали на четыре строки, поскольку в окне нужно оставить место под меню этажей.

Одновременно с формированием кнопок происходит наполнение массива NN номерами мест из поля GOST.MES. Массив заполняется, начиная со второго элемента. В дальнейшем это упростит ссылки на его элементы, поскольку первым GET-объектом в окне является скрытое POPUP-меню. Обработка выбора кнопок осуществляется в функции MS(), куда передается значение элемента массива под номером _Curobj, т.е. номер кнопки, на которой нажата клавиша Enter/Space. Функция MS() перемещает указатель записей на запись с совпадающим значением поля MES. Эта функция и функция TTT() полностью идентичны с одноименными функциями из программы GOST.PRG и здесь опущены.

Используя опцию WHEN команды BROWSE и (несколько модернизированную) функцию ET(), можно сделать так, чтобы не только BROWSE-окно отслеживало выбор пользователя в окне PLAN, но и наоборот, — перемещение курсора в окне GOST на запись, соответствующую другому этажу, вызывало предъявление плана этого этажа.

Хотя номера мест и поэтажные планы гостиниц изменяются редко, все же необходимо предусмотреть для пользователя возможность реагировать на эти события. Сам поэтажный план нарисовать несложно. Подразумевается что пользователи всех уровней должны уметь работать с текстовым редактором. Если пользователь привык к другому редактору, можно подключить его вместо внутреннего редактора FoxPro, указав имя редактора в файле CONFIG.FP. Можно предложить ему сделать план вообще вне FoxPro в виде текстового файла, который затем может быть импортирован в мемо-поле базы PLAN.DBF. Такую программу написать несложно. Этим мы заниматься не будем.

Более трудная задача — автоматизация заполнения полей базы X и Y, содержащих

координаты мест на плане этажа. Необходимо освободить пользователя от вычисления позиций курсора на плане этажа и от ввода их в базу. Решение этой проблемы приведено в программе PLAN.PRG.

Рабочий интерфейс состоит из двух окон. BROWSE-окно GOST1 отображает одно только поле номеров мест (MES) базы GOST.DBF. Окно REDPLAN предназначено для вывода плана этажа командой @...EDIT в режиме, не допускающем изменения. Поскольку в плане этажа могут устанавливаться пометки, в команде @...EDIT выводится не само поле PLAN.PLAN, а его копия (переменная M.PLAN).

Действия пользователя по занесению координат в поля X и Y сводятся к тому, что он выбирает в BROWSE-окне очередную запись с нужным номером места, а затем в окне, содержащем план этажа, устанавливает курсор под цифру-номер места и нажимает клавишу F3 или правую кнопку мыши. Обработка нажатия производится в процедуре NOM, куда передаются текущие оконные координаты курсора (ROW() и COL()). К ним прибавляется единица с тем, чтобы учесть, что нумерация строк/колонок в окне начинается с нуля, а нумерация строк в поле и символов в строке — с единицы. В соответствии с переданными координатами в переменной M.PLAN разыскивается видимый символ и запоминается в переменной Z1.

Чтобы предотвратить ошибки ввода, здесь предусмотрен ряд проверок. Сначала выясняется, из какого окна был сделан вызов. Процедура выполняется только в случае, если клавиша F3 была нажата в окне REDPLAN. Затем определяется, цифра ли Z1. Если да, то проверяются позиции справа и слева от выбранного места. Если рядом тоже цифра, значит, клавиша F3 была нажата, когда курсор находился не на номере места, а на номере этажа (у нас он трехзначный). Для этого значения смежных символов ранее были запомнены в переменных Z0 (символ слева) и Z2 (символ справа). И наконец, проверяется совпадение номера места с последней цифрой поля MES.GOST. Если все условия соблюдены, координаты курсора заносятся в поля Y.GOST и X.GOST, а справа от номера выбранного места появляется звездочка, указывающая пользователю, что координаты были им сохранены (m.plan=STUFF(...)). Перед этим прямоугольные экранные координаты звездочки (Y+1, X+2) преобразуются в номер символа внутри переменной M.PLAN и запоминаются в переменной L. С этой целью в цикле FOR сканируются все строки и их длины (плюс два символа завершения строки) суммируются в переменной L. При инициализации L уже учтено (L-M.X+2) — смещение по координате X. Если какое-либо из вышеперечисленных условий не выполняется (ELSE), выдается сообщение о неправильном позиционировании курсора.

С тем чтобы план этажа всегда соответствовал текущему номеру этажа, команда BROWSE снабжена опциями WHEN и VALID. При выходе из записи в VALID-функции VVV() запоминается номер этажа (первая цифра номера места) в переменной ETG. При входе в новую запись в WHEN-функции WWW() проверяется совпадение значения этой переменной и первой цифры следующей записи. Если они различны, значит, произошел переход на другой этаж и план должен быть обновлен. Для этого в базе PLAN.DBF разыскивается запись с совпадающим номером записи, из которой и считывается план нужного этажа.

*----Программа PLAN.PRG формирования значений полей Y и X базы-----
 *-----GOST.DBF. Нужны файлы GOST.DBF и PLAN.DBF-----

```
CLEAR
CLEAR MACRO
ON KEY LABEL f3 DO nom WITH ROW(),COL()
ON KEY LABEL f10 ACTIVATE WINDOW F10
ON KEY LABEL f2 ACTIVATE WINDOW redplan
ON KEY LABEL rightmouse DO nom WITH ROW(),COL()
DEFINE WINDOW gost1 FROM 1,1 TO 14,11 COLOR SCHEME 10
DEFINE WINDOW redplan FROM 5,25 TO 15,56 TITLE 'F2';
FOOTER 'Выбор места - F3/правая кнопка'
```

```
m.plan=' '
USE gost IN a
USE plan IN b
SELECT a
r=RECNO()
BROWSE TITLE 'F10';
FIELDS a.mes :H='Места' :P='9999',;
r=IIF(r=RECNO(1), '<', ' ') :H='';
WINDOW gost1 NOWAIT;
WHEN www() VALID :F vvv()
```

&& Вывод номеров мест


```

etg=LEFT(a.mes,1)
SELECT b
GO VAL(LEFT(a.mes,1)) IN b && Переход на запись, содержащую план
ACTIVATE WINDOW redplan
m.plan=b.plan
@ 0,0 EDIT m.plan SIZE 9,28 NOMODIFY          && Вывод плана этажа
READ CYCLE
DEACTIVATE WINDOW redplan,f10
ON KEY

PROCEDURE nom      &&-----Процедура обработки клавиши F3
PARAMETERS y,x
IF WONTOP()# 'REDPLAN'      && Если обращение к процедуре не из окна,
    RETURN                  && содержащего план - вызов из процедуры
ENDIF
* Находятся значения текущего (Z1) и смежных символов (Z0,Z2)
z1=SUBSTR(MLINE(m.plan,m.y+1),m.x+1,1)
z0=SUBSTR(MLINE(m.plan,m.y+1),m.x,1)
z2=SUBSTR(MLINE(m.plan,m.y+1),m.x+2,1)
* Если текущий символ цифра, а смежные - нет, и если номер
* места на плане совпадает с последней цифрой поля GOST.MES,
IF z1>='0' AND z1<='9' AND (z0<'0' OR z0>'9');
    AND (z2<'0' OR z2>'9') AND z1=RIGHT(a.mes,1)
    && в базе GOST.DBF запоминаются координаты курсора
    REPLACE a.y WITH m.y, a.x WITH m.x
    l=m.x+2      && Начальное значение переменной номера позиции
    FOR i=1 TO y && Сканирование строк переменной M.POLE
        l=l+LEN(MLINE(m.plan,i))+2 && Прибавление длины строки
    ENDFOR
    m.plan=STUFF(m.plan,l,1,'*') && Вывод звездочки
    SHOW GET m.plan      && Обновление окна REDPLAN
ELSE                    && Иначе выдется сообщение об ошибке
    WAIT 'Неверно установлен курсор' NOWAIT WINDOW
ENDIF
RETURN

FUNCTION www      &&-----Функция обновления курсора и плана этажа
r=RECNO(1)
IF etg#LEFT(a.mes,1)      && Если этаж изменился, в базе
    GO VAL(LEFT(a.mes,1)) IN b      && PLAN.DBF ищется нужная запись
    m.plan=b.plan      && откуда извлекается новый план
    SHOW GET m.plan
ENDIF
SHOW WINDOW F10 REFRESH
RETURN

FUNCTION vvv      &&-----Функция сохранения номера этажа
etg=LEFT(a.mes,1)
RETURN
*-----Конец модуля-----

```

Применять подобный интерфейс, конечно, имеет смысл только при наличии мыши. Хотя можно перемещаться здесь и не прибегая к ней, это непродуктивно.

В некоторых случаях может оказаться удобным использовать команду READ без команд GET. Такая "Общая" команда READ (Foundation READ) применяется для координации нескольких обычных команд READ при организации сложного интерфейса. Тогда она может управлять уже даже не отдельными окнами, а целыми "гнездами" пользовательских окон. Для контроля исполнения вложенных команд READ функция READKEY() была расширена таким образом, что стало возможным устанавливать причину завершения последней команды READ.

Глава 27. КОМАНДЫ ЯЗЫКА ЗАПРОСОВ SQL

В язык FoxPro включен ряд команд из языка запросов SQL (Structured Query Language). Он хорошо известен многим пользователям, работавшим на больших ЭВМ, и считается (почти?) стандартом. Хотя первоначально включение SQL в пакет FoxPro было вызвано желанием скорее придать формальную полноту средствам СУБД, чем потребностями программистов, рассматриваемые ниже команды являются полезным и быстрым инструментом обработки данных. Команды SQL могут непосредственно включаться в программы наряду с собственными командами FoxPro.

Создание баз данных

■ CREATE DBF <DBF-файл>

(<имя поля> <тип> [(<размер>[, <дробных разрядов>])
[, <имя поля>...]]) / FROM ARRAY <массив>

Команда создает новую базу данных <DBF-файл> с указанным именем. Для каждого поля задаются его имя, тип (одной из букв C, N, D, M, F, L), длина и число десятичных разрядов. Длина и точность не задаются для типов дата (D), логический (L) и примечаний (M), а точность — для символьного типа (C). Все требования к описанию полей базы данных — стандартные. Созданная база сразу открывается.

Описание полей может быть задано и из <массива> при использовании опции FROM ARRAY. Такой двумерный массив из четырех столбцов и строк в количестве, совпадающем с количеством полей, должен иметь указанное выше наполнение. Например, функция AFIELDS() как раз генерирует такой массив из открытой базы данных.

П р и м е р. Создание базы KADR.DBF.

```
CREATE DBF kadr (FAM C(25), TAB N(3),;  
DTR D, POL C(1), SEM C(1), DET N(1),;  
PODR C(15), SZAR N(7,2), PER M)
```

Не исключено, что такой способ покажется вам более удобным механизмом создания баз данных из программы (чем с помощью команды CREATE FROM). Команда допускает использование макроподстановки.

Дополнение базы

■ INSERT INTO <файл БД> [(<поле1> [, <поле2> [, ...]])] VALUES (<выр1> [, <выр2> [, ...]])

Команда добавляет записи в конец существующего <файла базы данных>, используя <выражения>, перечисленные после слова VALUES. Если опущены имена полей, <выражения> будут записываться в последовательные поля <базы данных> в соответствии с ее структурой.

Другая форма этой команды

■ INSERT INTO <имя БД> FROM ARRAY <массив> FROM MEMVAR

переносит данные, содержащиеся в указанном <массиве> (опция ARRAY), или данные из временных переменных (опция MEMVAR). Такие переменные должны существовать и иметь те же самые имена, что и поля базы данных (но с префиксом M). Такие переменные, например, вырабатываются по команде SCATTER MEMVAR. Поля, для которых нет переменных с подходящими именами, останутся пустыми.

Таким образом, команда INSERT соответствует паре команд — APPEND

BLANK и REPLACE.

В следующей команде база KADR.DBF дополняется новой записью с данными для полей FAM, TAB и SZAR.

```
INSERT INTO kadr (fam, tab, szar) VALUES ('Петров В.П.', 860, 5600)
```

Дополняемая база данных может быть и не открыта к моменту выполнения команды, однако после этого она останется открытой и активной.

Формирование запросов из базы данных

Команда является мощным средством обработки запросов. С ее помощью из базы-источника выделяются нужные данные и пересылаются на экран или в файл-приемник. Данные могут быть извлечены из разных баз, а также сгруппированы и упорядочены желаемым образом.

Команда имеет массу опций-возможностей. Ввиду этого сначала приведем ее предварительный синтаксис, который позволит затем лучше осознать детали.

- **SELECT** <что выводится>
FROM <откуда (источник)> **INTO** <куда (получатель)>
WHERE <каким условиям должно отвечать>
GROUP BY <колонки по которым выполняется группирование>
HAVING <условие группирования записей в одну строку>
ORDER BY <в каком порядке выводить данные>

Команда **SELECT** и вообще команды **SQL** мало зависят от текущего состояния среды FoxPro. Они сами открывают нужные им базы данных и индексные файлы. Если необходимых для выполнения команды индексов нет, они будут созданы, а по завершении команды уничтожены. Однако, конечно, лучше, чтобы применялись готовые индексы — для этого они должны быть открыты (используются только компактные индексы). Исключение составляют структурные **CDX**-индексы. Открытие соответствующей базы данных (например, с помощью команды **SELECT**) открывает и индекс.

Теперь рассмотрим ее более полный формат.

- **SELECT**
[DISTINCT] [**<псевдоним>.**]**<выражение>** [**AS** **<колонка>**]
[, [<псевдоним>.**]**<выражение>** [**AS** **<колонка>**]...]**
FROM **<БД>** [**, <БД>** ...]
[[INTO** **<получатель>**]/**TO FILE** **<файл>****
[ADDITIVE**]/**TO PRINTER**]]**
[NOCONSOLE**] [**PLAIN**] [**NOWAIT**]**
[WHERE** **<условие связи>****
[AND** **<условие связи>**...]**
[AND/OR** **<условие отбора>****
[AND/OR** **<условие отбора>**...]]]**
[GROUP BY** **<колонка>** [**, <колонка>** ...]]**
[HAVING** **<условие отбора>**]**
[ORDER BY** **<колонка>** [**ASC/DESC**] [**, <колонка>** [**ASC/DESC**]...]]**

Термин "колонка" здесь очень близок к понятию "поле базы" данных, но может быть и выражением. Кроме того, вследствие выборки мы можем получить как новую базу данных, так и текстовый файл или даже только отображение на экране, т.е. колонки.

Команда **SELECT** допускает включение в себя других внутренних команд **SELECT** (формирование подзапросов).

Все примеры использования команды **SELECT** сгруппированы в конце

раздела. Сейчас рассмотрим опции команды.

Указание результатов выборки и источников данных.

```
SELECT [DISTINCT] [<псевдоним>.]<выражение> [AS <колонка>]  
FROM <БД> [<псевдоним>] [,<БД> [<псевдоним>] ...]
```

Здесь указывается, что и откуда берется в выборку. Перед словом FROM перечисляются отбираемые <выражения>, а после — имена баз, из которых берутся данные.

<Выражение> может быть полем записи из БД, константой (выводимой в каждой строке выборки), функцией (в том числе и ПФ) от переменных, полей и т.п. Если <выражение> является именем поля, то оно может быть составным (с включением имени базы данных или псевдонима), в особенности если выборка делается из нескольких баз, где имена полей совпадают, например поля TAB из баз KADR.DBF и BRIG1.DBF. Псевдонимом может быть не только "официальный" псевдоним (ALIAS) базы, но и любое другое имя, которое вы ей присвоите в команде SELECT. Это задаваемое временное имя указывается в опции <псевдоним> после слова FROM. Никаких последствий за пределами команды SELECT такое назначение не имеет.

Если необходимо построить выборку из всех полей базы, вместо их перечня можно указать символ "*".

В результате выполненной выборки получается совокупность колонок, заголовками которых могут быть имена полей. Если имена совпадают, то такие колонки получают совпадающие имена, к которым присоединяется одна из букв (по алфавиту), например TAB_A, TAB_B и т.д. Аналогичным образом даются имена колонкам, полученным в результате вычисления выражений. Их имена состоят из слова EXP и последовательных чисел. Так, возможны имена EXP_1, EXP_2 и т.д. Исключения составляют выражения, использующие собственные функции SQL, например функции MIN(), MAX() и др. Имена колонок в этом случае будут включать имена функций.

Если нас не устраивают имена, формируемые по умолчанию, можно назначить свои, указав их после слова AS в виде

```
<выражение> AS <новое имя колонки>
```

Такое переименование имеет смысл, если выборка помещается во вновь создаваемую базу данных. Имя дается по правилам, принятым в FoxPro, и должно быть задано латинскими символами.

В <выражении> могут быть использованы любые функции FoxPro. Кроме того, здесь есть еще собственные специальные арифметические функции, действующие "по вертикали". Это функции вычисления среднего, минимального и максимального значений, суммирования, а также количества записей:

```
AVG(<выр>), MIN(<выр>), MAX(<выр>), SUM(<выр>), COUNT(<выр>).
```

Последняя функция может иметь в качестве аргумента звездочку (COUNT(*)), что означает подсчет всех записей, попавших в выборку.

Включение опции DISTINCT исключает возможность вывода одинаковых строк в выборке.

Указание объекта, куда пересылается выборка.

Следующие опции задают "получатель" данных выборки. Им может быть база данных, массив, текстовый файл, экран и принтер.

Кроме того, информация может быть переслана в так называемый Курсор (будем писать с большой буквы). Курсор — это временный набор данных, который может быть областью памяти или временным файлом FoxPro (этот процесс от нас не зависит) и имеет режим "Только чтение". Данные Курсора

могут быть, например, предъявлены в команде BROWSE, напечатаны, из них может быть образовано меню и т.д. Курсор может быть обработан другой командой SELECT. К колонкам Курсора надо обращаться по имени этих колонок, возможно, с префиксом — именем Курсора (через точку).

Итак:

INTO <получатель>

<Получатель> может быть одного из следующих типов:

- ARRAY <массив> — задается вновь создаваемый двумерный <массив>
- CURSOR <курсор> — задается имя Курсора.
- DBF/TABLE <БД> — новая база данных с указанным именем. Слова DBF и TABLE здесь являются синонимами.

Кроме того, данные можно переслать в файл или на принтер.

TO FILE <файл> [ADDITIVE]/TO PRINTER — выборка посылается в текстовый <файл> или на принтер. Если используется слово ADDITIVE, то выборка будет добавлена в конец существующего файла без его перезаписи.

Следующие опции имеют смысл только при выдаче на экран (команда используется без слова INTO):

NOCONSOLE — выборка не выдается на экран.

PLAIN — заголовки колонок не выдаются.

NOWAIT — не делаются паузы при заполнении экрана.

Критерий отбора данных

WHERE <условие связи> [AND <условие связи> ...]

[AND/OR <условие отбора> [AND/OR <условие отбора>...]]

Здесь:

<Условие связи> — применяется в случае, если выборка делается более, чем из одной базы данных, и указывает критерий, которому должны отвечать поля из разных баз. В условии связи указываются поля из разных баз. Здесь разрешается использовать знаки отношения =, #, ==, >, >=, <, <=. Допускается задание нескольких критериев, соединенных знаком AND.

<Условие отбора> — строится аналогично, но из выражений только для одной базы, и допускается использование логических операторов OR и NOT.

Условия, кроме любых функций FoxPro, могут содержать следующие операторы SQL:

LIKE — позволяет построить условие сравнения по шаблону, где символ "_" указывает единичный неопределенный символ в строке, "%" — любое их количество. Эти символы аналогичны символам маски "?" и "*" в MS DOS. *Формат оператора:*

<выражение> LIKE <шаблон>

BETWEEN — проверяет, находится ли выражение в указанном диапазоне. *Формат оператора:*

<выражение> BETWEEN <нижнее знач.> AND <верхнее знач.>

IN — проверяет, находится ли выражение, стоящее слева от слова IN, среди перечисленных справа от него (аналогично функции INLIST).

Формат оператора:

<выражение> IN (<выражение>, <выражение>, ...)

Все указанные операторы можно комбинировать с помощью связок OR, AND, NOT и скобок. Операторы LIKE и BETWEEN не следует путать с одноименными функциями FoxPro, которые, впрочем, тоже можно использовать.

Группирование данных.

GROUP BY <колонка>[,<колонка>...] — задаются колонки, по которым производится группирование выходных данных. Все записи базы, для которых значения колонок совпадают, отображаются в выборке единственной строкой. Группирование удобно для получения некоторых сводных характеристик (сумм, количеств) группы.

HAVING <условие отбора> — опция задает критерий отбора данных в каждую сформированную в процессе выборки группу.

Сортировка

ORDER BY <колонка> [ASC/DESC] [,<колонка> [ASC/DESC]...] — опция задает упорядочение по заданной колонке/колонкам. По умолчанию сортировка выполняется по возрастанию (ASC), но может быть задана и по убыванию (DESC).

Примеры запросов

1. Выборка всех полей из базы KADR.DBF. Все колонки выборки будут иметь имена полей базы данных.

```
SELECT * FROM kadr
```

2. Вывод минимального, максимального и среднего значений поля SZAR (средняя зарплата). Колонки получают имена MIN_SZAR, MAX_SZAR и AVG_SZAR.

```
SELECT MIN(szar),MAX(szar),AVG(szar) FROM kadr
```

3. Вывод фамилий работников, получающих от 3000 до 8000 рублей.

```
SELECT fam FROM kadr WHERE szar BETWEEN 1000 AND 3000
```

Вывод фамилий всех сотрудников, кроме работающих в подразделениях ОГМ и КБ.

```
SELECT fam FROM kadr WHERE podr NOT IN ('ОГМ','КБ')
```

4. Выборка названий всех подразделений (поле PODR) предприятия из базы KADR.DBF. Опция DISINST предотвращает повторный вывод одних и тех же названий, если они повторяются.

```
SELECT DISTINCT podr FROM kadr
```

5. Выборка фамилий (FAM) всех мужчин из KADR.DBF.

```
SELECT fam FROM kadr WHERE pol='М'
```

6. Выборка всех фамилий и табельных номеров из KADR.DBF, сцепленных с выработками из базы BRIG1.DBF для записей, у которых совпадают табельные номера.

```
SELECT s.fam,s.tab,t.tab,t.vir;  
FROM kadr s,brig1 t WHERE s.tab=t.tab
```

Здесь для сокращения записи команды базам KADR.DBF и BRIG1.DBF заданы новые временные имена S и T. Они никак не связаны с рабочими областями. Базы будут открыты в свободных областях системы. Сами колонки выборки получают имена FAM, TAB_A, TAB_B, VIR.

7. Если мы хотим задать собственные имена колонкам, а не использовать умолчания, нужно воспользоваться опцией AS.

Пусть нужно вывести фамилии и табельные номера (поля FAM и TAB) по алфавиту и с другими именами колонок FAMILII и TABEL.


```
SELECT fam AS фамилии, tab AS табел FROM kadr ORDER BY fam
```

8. Выборка фамилий всех родившихся в текущем месяце с указанием дня (числа) рождения, количества лет и премии по этому поводу — 50 % от значения средней зарплаты.

```
SELECT fam,DAY(dtr),'число',YEAR(DATE())-YEAR(dtr),'лет',;  
'премия',0.5*szar FROM kadr WHERE MONTH(dtr)=MONTH(DATE())
```

Колонки получают имена FAM и от EXP_2 по EXP_7.

9. Вывод полей FAM и TAB, отсортированных по полям POL (главное поле) и FAM (подчиненное поле) в базу FAMTAB.DBF, которая затем открывается в текущей области.

```
SELECT fam,tab FROM kadr ORDER BY pol,fam INTO TABL famtab
```

Чтобы увидеть содержимое этой базы, можно, например, сразу ввести команду BROWSE.

10. Вывод для каждого табельного номера из базы KADR.DBF выработок из баз BRIG1.DBF и BRIG3.DBF, а также суммарной выработки работника в обеих бригадах.

```
SELECT kadr.tab,brig1.vir,brig3.vir,brig1.vir+brig3.vir;  
FROM kadr,brig1,brig3;  
WHERE kadr.tab=brig1.tab AND kadr.tab=brig3.tab
```

Имеется в виду, что работник может работать в нескольких бригадах в течение месяца, но в каждом бригадном файле он может встретиться только раз.

11. Вывод фамилий всех работников, работавших ранее в конструкторском бюро (КБ). Поиск ведется в мемо-поле PER базы KADR.DBF.

```
SELECT fam FROM kadr WHERE per LIKE "%КБ%"
```

12. Вывод табельных номеров и суммарной выработки каждого работника в бригаде номер 1. Вывод осуществляется в порядке увеличения табельных номеров.

```
SELECT tab,SUM(vir) FROM brig1 GROUP BY tab ORDER BY tab
```

Задача имеет смысл, если один и тот же человек может встречаться несколько раз в одной бригаде, например если в бригадном файле фиксируются не итоговые выработки, а все наряды.

13. Вывод названий всех подразделений, количества сотрудников и значений суммарной заработной платы (фонда оплаты). Информация выводится только для подразделений, где количество сотрудников больше пяти.

```
SELECT podr,COUNT(*),SUM(szar);  
FROM kadr GROUP BY podr HAVING COUNT(*)>5
```

Все приведенные примеры легко проверить, набрав указанные команды прямо в командном окне. Нужно только, чтобы существовали используемые базы данных.

В заключение отметим, что для более детального знакомства с возможностями команды SELECT и языка SQL следует обратиться к документации по системе FoxPro и литературе [3], где описан SQL.

Глава 28. СИСТЕМНЫЙ ИНТЕРФЕЙС FoxPro

Обратимся к системному интерфейсу СУБД в той мере, в какой он полезен программисту. Рассмотрим основное меню системы и средства автоматизированного создания приложений (Генераторы отчетов, экранов, меню и Менеджер проектов). Остановимся только на важнейших функциях интерфейса, опустив некоторые подробности. С целью максимальной ясности изложения и экономии места в экранах иногда не отображаются несущественные детали и очевидные элементы, а также вносятся некоторые "косметические" изменения.

Многие из пунктов основного и других меню подразумевают возможность обращения к средствам управления следующего, более низкого уровня, которые могут быть окнами ввода, меню или кнопками. В FoxPro такие пункты завершаются многоточием и вызов, который они осуществляют, определяется термином "вызов диалога".

Главное меню СУБД. Сразу после загрузки системы на экране предъявляется главное меню СУБД, через которое мы можем непосредственно управлять данными, писать и отлаживать программы, а также настраивать рабочую среду системы. Хотя это меню и вся интерактивная среда FoxPro ориентированы, скорее, на пользователя, чем на программиста, глупо было бы ее игнорировать поскольку они содержат множество полезных средств, позволяющих существенно облегчить труд разработчика. Вид системного меню приведен на рис.28.1.

Активация строки главного меню осуществляется нажатием клавиши F10 или Alt. Непосредственный вызов нужного пункта меню может быть осуществлен и нажатием клавиш Alt и выделенной цветом ("горячей") буквы пункта (например, Alt-S вызовет главное меню сразу со вспомогательным меню System). Далее такое POPUP-меню второго уровня в тексте будем называть по имени PAD-пункта главного меню, например System-меню. В зависимости от режима работы пользователя справа от линейки главного меню появляются некоторые дополнительные меню следующего уровня.

При вызове меню может оказаться, что некоторые его элементы окрашены в иной ("приглушенный") цвет. Это значит, что они не могут быть выбраны в данный момент (курсор не фиксируется на них) ввиду того, что нет соответствующего объекта. Например, невозможно просмотреть базу данных, если она еще не открыта. Рассмотрим функции системного меню FoxPro.

Общесистемные функции (SYSTEM-меню). Здесь реализованы средства доступа к файлам, Help, а также "Настольная оргтехника" — календарь, калькулятор и т.д.:

About FoxPro	— предъявление фирменного номера пакета;
Help...	— справочная информация (HELP-системы);
Macros...	— закрепление за функциональными клавишами заданных макропоследовательностей;
Filer	— вызов файлового процессора;
Calculator	— вызов калькулятора;
Calendar/Diary	— вызов календаря-дневника;
Special Charact	— таблица символов псевдографики;
ASCII Chart	— таблица ASCII-кодов;
Capture	— взятие в буфер заданной области экрана;
Puzzle	— игра-головоломка "Пятнадцать".

System	File	Edit	Database	Record	Program	Window
About FoxPro Help... F1 Macros...	New... Open... Close	Undo ^U Redo ^R	Setup... Browse	Append Change	Do... ^D Cancel Resume ^M	Hide Clear
Filer Calculator Calendar/Diary Special Characters ASCII Chart Capture Puzzle	Save Save as... Revert	Cut ^X Copy ^C Paste ^V Clear	Append From.. Copy To... Sort... Total...	Goto... Locate... Continue ^K Seek...	Compile... Generate FoxDoc FoxGraph...	Move ^F7 Size ^F8 Zoom ↑ ^F10 Zoom ↓ ^F9 Cycle ^F1 Color...
	Printer Setup Print...	Select All ^A	Average... Count... Sum... Calculate... Report... Label...	Replace... Delete... Recall...	Do <PRG-файл> ^O	Command ^F2 Debug Trace View
	Quit	Goto Line... Find... ^F Find Again ^G Replace And find Again ^E Replace All Preferences...	Pack Reindex			0 KADR 1 KADR.PRG

Рис.28.1. Главное системное меню СУБД FoxPro-2.0

Исключительно полезным для программиста является доступ к справочной информации через это меню или просто с помощью клавиши F1. В FoxPro Help очень богатый и удобный инструмент изучения языка и получения различных справок.

При вызове подсказки сначала мы попадаем в ее оглавление, где перечислены все команды и функции, а также другая важная информация. Выбор любого его пункта выводит в соответствующий раздел документации (рис.28.2).

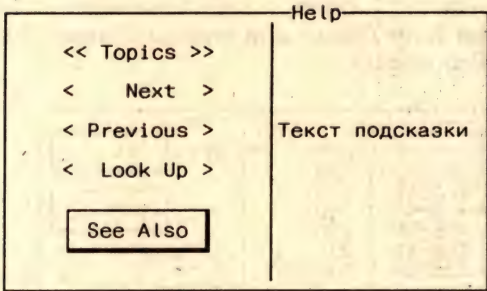


Рис.28.2

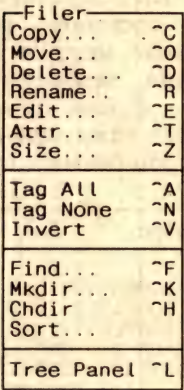


Рис.28.3

Окно Help кроме содержательной части содержит ряд управляющих средств. Кнопки Topics, Next и Previous переносят читателя назад в оглавление или к следующему/предыдущему разделу помощи. Через POPUP-меню See Also можно сразу перейти к разделам, на которые есть ссылки в данном месте. Кнопка Look Up обычно "приглушена". Она становится доступной, если вы выделили какой-нибудь текст внутри данного экрана с помощью мыши или клавиатуры. Теперь, если обратиться к кнопке Look Up, система будет искать раздел помощи с заданной командой/функцией. К Help можно обращаться практически из любого места системы. Если в тексте, например программы, вами выделено какое-то слово, то нажатие клавиши F1 сразу выведет в нужный раздел Help (если есть).

Help допускает взятие из него в буфер компьютера любого фрагмента текста (здесь действуют клавиши редактора Ctrl-C), в том числе и примеров программ. Далее они могут быть помещены в командный файл и запущены на исполнение. Это облегчает изучение языка FoxPro и вообще проведение различных программных "экспериментов".

Полезным для программиста является и пункт Filer, через который он попадет в файловый монитор FoxPro, или просто Файлер. Рядом с главным меню появляются специальный пункт Filer и связанное с ним FILER-меню, изображенное на рис.28.3. Оно полезно прежде всего тем, что содержит указания на "горячие" клавиши для быстрого выбора кнопок управления в окнах Файлера при работе без мыши. Смысл пунктов меню станет ясен из описания этих окон.

В основном окне Файлера (рис.28.4) предьявляется содержимое текущей директории.

Для всех файлов здесь указаны: имя, расширение имени, размер, дата и время последней модификации файла и его атрибуты. Находясь в списке файлов, мы можем перемещаться по нему, а также переходить в другие

директории через пункты [...]/.]. В правом углу через кнопку DRV можно перейти на любой дисковод, а через кнопку DIR — в любую старшую директорию. Кроме того, здесь же мы можем указать любую маску, ограничивающую имена предъявляемых файлов. На рис.28.4 — все имеющиеся файлы (Files Like *.*). Внизу и справа от списка файлов изображены кнопки, посредством которых можно осуществлять файловые операции — поиска <Find>, копирования <Copy>, перемещения <Move>, удаления <Delete>, сортировки <Sort>, редактирования <Edit>, изменения атрибутов <Attr>, переименования <Rename> и установления размера <Size>. При этом системой по необходимости будут предлагаться другие вспомогательные меню и диалоги. Для реализации перечисленных действий предварительно необходимо выбрать обрабатываемые файлы/директории. Отобранные файлы помечаются слева треугольником. Пометка осуществляется нажатием клавиши Enter/Space или кнопки мыши. Повторное нажатие клавиш снимает сделанную пометку.

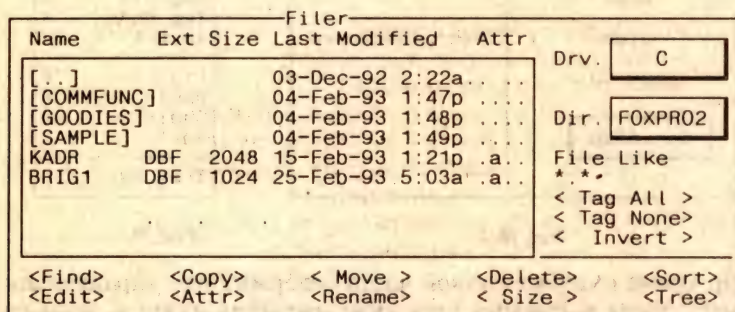


Рис.28.4

Выбор другого файла также снимает предыдущую пометку. Однако выбрать можно и несколько файлов. Для этого после пометки одного файла, следует нажать и удерживать клавишу Shift и с помощью клавиш со стрелками переместить курсор. Все пройденные им файлы будут помечены. Если нужно выбрать несмежные файлы, следует отпустить клавишу Shift, перейти в нужное место, снова нажать ее и пометить файл клавишей Enter/Space. Частичное снятие пометок выполняется совершенно аналогичным образом. Полное снятие пометок может быть выполнено выбором любого файла без удержания клавиши Shift. Кроме того, кнопками <Tag All>, <Tag None>, <Invert> можно осуществить выбор всех файлов в директории, снять все сделанные пометки и сделать инверсию пометок. В последнем случае все помеченные файлы станут непомеченными и наоборот.

Через кнопку <Tree> можно перейти к древовидному представлению данных (рис.28.5). Действие кнопок здесь аналогично предыдущему, но только по отношению не к файлам, а к директориям. Кнопками <Chdir> и <Mkdir> можно перейти в иную директорию или создать новую. Текущая директория отмечается точкой.

Через пункт Special Characters в SYSTEM-меню можно получить быстрый доступ к специальным символам расширенной ASCII-таблицы. Такая возможность очень полезна при создании различных рисунков из символов псевдографики. Для этого сначала необходимо удобным образом разместить на экране таблицу символов и окно редактора. Далее с помощью клавиш или мыши можно быстро перемещать курсор в обоих окнах, выбирая нужный символ и место его расположения.

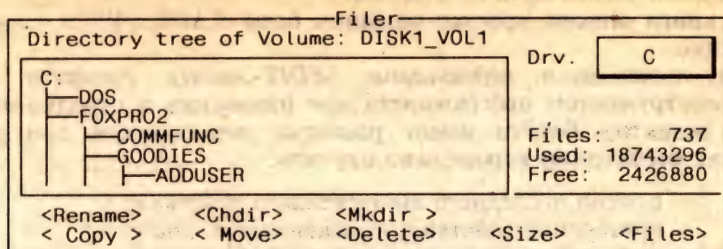


Рис.28.5

Работа с файлами СУБД (FILE-меню). В этом меню находятся средства управления (открытие, закрытие, создание и т.п.) файлами, принадлежащими только FoxPro:

New	— создание нового файла;
Open	— открытие имеющегося файла;
Close	— закрытие активного окна;
Save	— сохранение файла с тем же именем;
Save as	— сохранение файла с новым именем;
Revert	— выдача предыдущей версии текстового файла до всех изменений;
Printer Setup	— определение принтера/порта для выдачи файла, форматирование листа;
Print	— печать файла;
Quit	— выход в ДОС.

Здесь важным, в особенности в первоначальный момент, являются первые два пункта меню, через которые можно создавать новые (New) или открывать (Open) уже имеющиеся файлы всех основных типов в СУБД. При выборе пункта New пользователю предлагается меню, изображенное на рис.28.6, в котором сверху вниз перечислены все основные типы файлов, а именно базы данных, программные, текстовые, индексные, файлы генератора отчетов, почтовых наклеек, экранов, меню, запросов и менеджера проектов.

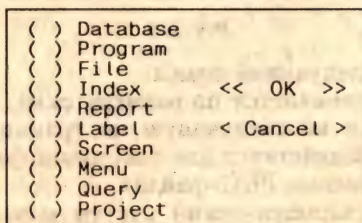


Рис.28.6

Пункт Open устроен похоже, но кроме типов файлов здесь сразу открывается содержимое директории, в которой можно выбрать нужный файл. Вообще при начальном изучении языка FoxPro, пока вами еще не освоены необходимые команды, целесообразно обращаться к средствам среды пользователя. Следует поискать на диске файлы определенных типов (их примеры в большом количестве имеются в пакете), запустить на выполнение (можно через меню Program), изучить полученный результат и по

возможности исследовать сами программы.

В начале книги описан процесс создания базы KADR.DBF и сопровождающие его меню.

Работа с текстовым редактором (EDIT-меню). Редактор является важнейшим инструментом программиста при написании и отладке программ. Встроенный редактор FoxPro имеет развитые возможности при работе с текстами, и их необходимо хорошенько изучить:

- Undo — отмена последнего выполненного действия;
- Redo — повторение действия, отмененного в Undo;
- Cut — удаление из текста выделенной области с взятием ее в буфер клавиатуры;
- Copy — копирование выделенного текста в буфер;
- Paste — вывод содержимого буфера в текст;
- Clear — очистка выделенной области без взятия ее в буфер;
- Select All — выделение всего текста окна;
- Goto Line — переход к строке с указанным номером;
- Find — поиск заданного фрагмента текста;
- Find Again — поиск следующего вхождения текста;
- Replace And find Again — последовательный поиск вхождений с заменой;
- Replace All — полная замена всех вхождений текста;
- Preferences — настройка редактора.

Важнейшие клавиши и функции редактора указаны в гл.11. Здесь мы только рассмотрим полезные для программиста настройки редактора (Preferences), которые устанавливаются в специальном окне (рис.28.7).

<input type="checkbox"/> Wrap words	Tab size: 4
<input checked="" type="checkbox"/> Auto indent	
<input checked="" type="checkbox"/> Make backup	<input type="checkbox"/> Use these ...
<input type="checkbox"/> Add line feeds	
<input type="checkbox"/> Status line	
<input type="checkbox"/> Compile when saved	
<input type="checkbox"/> Add Ctrl-Z	<input type="checkbox"/> Save preferences
<input type="radio"/> Left justify << OK >>	
<input type="radio"/> Right justify	
<input type="radio"/> Center justify < Cancel >	

Рис.28.7

Его установки имеют следующий смысл:

Wrap words — текст выравнивается по размеру окна. Длинная строка автоматически переносится на следующую по границе слова. По умолчанию ([X]) выравнивание действует для текстовых файлов и мемо-полей и не действует в программных PRG-файлах.

Auto indent — создает автоматический отступ новой строки (при нажатии Enter), равный отступу предыдущей строки.

Make backup — сохранение файла с созданием ВАК-копии.

Add line feeds — при сохранении файла все строки завершаются символом возврата каретки.

Status line — в верхней части экрана редактора отображается статус-строка, где можно видеть номер текущей колонки/строки.

Compile when saved — установка действует только для программных PRG-файлов и вызывает автоматическую компиляцию отредактированных программ.

Add Ctrl-Z — при сохранении в файле текст дополняется символом Ctrl-Z.
 Left justify — выравнивание текста влево.
 Right justify — выравнивание текста вправо.
 Center justify — центрирование текста.
 Tab size — размер отступа табуляции.
 Use these preferences ... — использование установленной настройки (по умолчанию) для файлов с заданным расширением и мемо-полей.
 Save preferences — то же, но для файла с конкретным именем.
 Все настройки как редактора, так и других компонентов системы сохраняются в ресурсном файле FoxPro (обычно в файле FOXUSER.DBF).

Работа с базой данных. DATABASE-меню содержит исключительно средства управления базой данных и связанными с ней файлами:

- Setup — модификация/создание/открытие баз данных, а также индексов и фильтров;
- Browse — вызов команды полноэкранного редактирования, при этом главное меню дополняется справа пунктом BROWSE;
- Append From — добавление записей из других БД/файлов;
- Copy To — копирование в другие БД/файлы;
- Sort — сортировка записей;
- Total — создание БД с суммами по заданному признаку;
- Average — среднее арифметическое числовых полей;
- Count — подсчет записей с указанным признаком;
- Sum — суммирование числовых полей;
- Calculate — вычисления в базе данных;
- Report — подключение ранее созданной формы отчета;
- Label — подключение ранее созданной формы почтовых наклеек;
- Pack — физическое удаление помеченных записей;
- Reindex — переиндексация базы;

Выбор пункта Browse вызовет предъявление данных, поля которых расположены горизонтально, и меню настройки режима. Это меню третьего уровня мы также рассмотрим здесь (рис.28.8) ввиду его важности.

Browse		
Browse/Change/Append		- поля отображены горизонтально (Browse), вертикально (Change) или в режиме дополнения (Append)
Grid On/Off		- вертикальные разделители полей Есть/Нет
Link/Unlink		- синхронное/асинхронное вертикальное
Partitions		- перемещение курсора в разделенных окнах
Change Partition	^H	- переход в другую часть разделенного окна
Size Field		- установка видимого размера поля
Move Field		- перемещение поля относительно других
Resize Partitions		- разделение окна на две части
Goto...		- переход к указанной записи
Seek...		- индексный поиск
Toggle Delete	^T	- пометка записи к удалению
Append Record	^N	- дополнение базы новой записью

Рис.28.8

Работа с записями базы данных (RECORD-меню). Меню реализует действия по обработке базы данных:

- Append — вызов окна дополнения базы новыми записями;
- Change — вызов окна редактирования записей;

- Goto — переход к указанной записи;
- Locate — последовательный поиск записи по ключу;
- Continue — продолжение поиска (поиск следующей записи);
- Seek — ускоренный индексный поиск по ключу;
- Replace — изменения (вычисления) в базе данных;
- Delete — пометка записей, предназначенных для удаления;
- Recall — снятие пометок для удаления.

Работа с командными файлами (PROGRAM-меню). Через это меню можно управлять исполнением командных файлов (типа PRG и других), а также компиляцией и генерированием приложений:

- Do — выбор командного файла и исполнение программы;
- Cancel — прерывание программы;
- Resume — продолжение программы;
- Compile — переход в окно компиляции программы;
- Generate — вызов генератора приложений;
- FoxDoc — вызов документатора программ;
- FoxGraph — вызов графического пакета FoxGraph;
- Do <PRG-файл> — исполнение текущей программы (<PRG-файла>), загруженного во внутренний редактор FoxPro.

Последний пункт меню появляется только при работе с редактором, вызванным по команде MODIFY COMMAND <файл>. При нажатии клавиш Ctrl-O запрашивается необходимость сохранения текущего файла, однако окно редактора остается на экране. Это полезная возможность, но реализована она не слишком удобно. Другую реализацию такой функции мы рассмотрим в гл.31.

Документатор позволяет получать тщательно разобранные и снабженные комментариями тексты программ.

Использование пункта Generate будет рассмотрено позже.

Работа с окнами (WINDOW-меню). Здесь сосредоточены средства управления системными и пользовательскими окнами FoxPro: открытие, закрытие, перемещение, изменение размера и т.д., а также средства отладки готовых программ:

- Hide — окно удаляется с экрана, но не из памяти;
- Clear — очистка текущего окна;
- Move — перемещение окна;
- Size — изменение размера окна;
- Zoom — увеличение окна во весь экран;
- Zoom — стягивание окна в одну строку-заголовок;
- Cycle — последовательное циклическое переключение окон активных окон;
- Color — установление цвет объектов FoxPro;
- Command — переход в командное окно;
- Debug — вызов отладочного окна Debug;
- Trace — вызов окна Trace для трассировки программ;
- View — переход в окно VIEW управления режимами работы с базами данных;
- 0 KADR — переход в окно 0 с файлом KADR.DBF (пример);
- 1 KADR.PRG — переход в окно 1 с файлом KADR.PRG (пример).

Системное меню оболочки FoxPro является объектом, доступным для настройки. Его конфигурация может быть адаптирована пользователем по

своему усмотрению. Меню может быть радикально перекомпоновано, из него возможно изъятие каких-то элементов и дополнение новыми. Очень легко при желании сделать надписи в нем по-русски. Системное меню можно использовать и в программах. Более того, оно (в несколько усеченном виде) по умолчанию присутствует в готовых прикладных системах в любых их формах (FXP-, APP-, EXE-файлах) и может быть вызвано клавишами F10/Alt.

Следующая команда управляет доступом к системному горизонтальному BAR-меню и его компонентам.

- SET SYSMENU ON/OFF/AUTOMATIC/TO [DEFAULT]
/TO [<список системных POPUP-меню>
/<список системных PAD-пунктов>]

Опции команды:

ON — меню доступно, но невидимо. Его можно вызвать клавишами Alt и F10 при ожидании ввода с клавиатуры (команды: BROWSE, READ, MODIFY COMMAND и т.п.). Принято по умолчанию.

OFF — меню недоступно.

AUTOMATIC — меню доступно и постоянно присутствует на экране.

TO DEFAULT — восстановление исходного вида системного меню.

Опции TO <список POPUP-меню/PAD-пунктов> указывают, какие именно доступны POPUP-меню и элементы горизонтального меню.

Устройство системного меню можно изучить, обратившись к генератору меню, а в нем — к пункту Quick Menu. Имена всех элементов системного меню (они перечислены в Приложении) могут быть также получены с помощью функции

- SYS(2013)

При создании прикладных систем возникает естественное желание отключить системное меню (SET SYSMENU OFF). Следует учитывать, однако, что при этом одновременно утрачиваются важные клавишные комбинации, управляющие конфигурацией BROWSE-окна (Ctrl-N, Ctrl-T) и пользовательских окон (Ctrl-F9, Ctrl-F10, Ctrl-F7, Ctrl-F9), а также облегчающие работу при редактировании данных как в базе, так и в окне редактора (Ctrl-C, Ctrl-X, Ctrl-V, клавиши выделения текста). Хотя большинство из выполняемых ими действий могут быть запрограммированы, во многих случаях целесообразно оставить системное меню, возможно, изъяв из него ненужные пункты. Конфигурация системного меню может быть установлена путем перечисления после слова TO через запятую имен необходимых подчиненных POPUP-меню.

Глава 29. ГЕНЕРАТОРЫ ПРИЛОЖЕНИЙ

В СУБД имеются развитые генераторы приложений, позволяющие быстро создавать, почти не прибегая к непосредственному программированию, некоторые заготовки. Это, в частности, генераторы отчетов, экранов и меню. Здесь программист во многом освобожден от рутинного и утомительного труда по расчету положений видимых объектов (текстов, полей, переменных, рамок, меню) на экране/принтере. Все проектирование теперь сводится к физическому размещению нужных элементов в специальном окне проектирования — планшете, облик которого будет полностью соответствовать будущему виду экрана/отчета.

В результате работы генератора формируются файлы, хранящие образы созданных отчетов, экранов, меню, проектов и т.п. Это обычные файлы БД с файлами примечаний, но с расширениями имен не DBF и FPT, а с другими расширениями (FRX/FRT, SCX/SCT, MNX/MNT и т.д.). При желании такие базы могут быть открыты по команде USE <файл> и просмотрены, например, по команде BROWSE. Расширение в таком случае указывать обязательно. Однако никакое непосредственное их редактирование не допускается.

Работа в планшете. Объектом на планшете являются любая созданная надпись, поле, рамка и их множества. При работе с планшетом следует только освоить простую технику манипулирования объектами, которая сводится к следующему.

Создание объекта

Установить курсор и выбрать тип объекта из меню генератора или нажать нужную комбинацию клавиш (см. пункты меню генератора).

Перемещение среди объектов

С помощью клавиш Tab/Shift-Tab — в порядке следования объектов. В произвольном порядке — с помощью клавиш управления курсором или мышью.

Выбор объекта

Установить курсор на объект и нажать клавишу Space или кнопку мыши. Объект окрасится контрастным (красным) цветом. Выбор другого объекта отменяет предыдущий выбор. Выделенные объекты можно перемещать, удалять, копировать.

Выбор множества объектов

Установить курсор в незанятое объектами пространство экрана и нажать клавишу Space. Появится изображение точки. С помощью клавиш со стрелками переместить курсор таким образом, чтобы очертить точками нужную область экрана и нажать клавишу Space или Enter. Здесь удобно использовать мышь. Для этого в начальной точке нужно нажать кнопку мыши, а в конечной — ее отпустить. При этом будут выделены все объекты, попавшие (хотя бы частично) в очерченную область.

Можно сделать множественный выбор и следующим образом. Сначала выделяется один из нужных объектов. Далее при нажатой клавише Shift курсор перемещается к следующему объекту, который выделяется клавишей Space и т.д. Идентичный эффект имеет (при удерживаемой клавише Shift) нажатие кнопки мыши.

Аналогичным образом делается отмена выделения некоторых объектов из множества.

Полная отмена выбора

Нажать клавишу Space/Enter или кнопку мыши. Курсор должен находиться не на объекте.

Перемещение объекта

Выбранный объект/объекты буксируются мышью или клавишами со стрелками. В последнем случае в завершение нажать клавиши Enter/Space.

Изменение размеров

Нажать клавиши Ctrl-мышь или Ctrl-Space, когда курсор находится на объекте. Объект будет выделен мерцанием. Далее с помощью клавиш со стрелками или мышью можно изменить размер выделенного объекта. Если мышь не используется, в заключение следует нажать Enter/Space.

Модификация объекта

Переместиться на объект и нажать Enter или дважды кнопку мыши. При этом будет развернуто окно диалога, в котором можно осуществить содержательное изменение объекта.

Удаление объекта/объектов

После выделения объекта нажать Ctrl-X. Вообще здесь действуют функции редактора FoxPro. При нажатии Ctrl-X или Ctrl-C происходит взятие объекта в буфер клавиатуры (при Ctrl-C — без удаления с экрана). Далее, воспользовавшись буфером, можно выполнить копирование.

Выделенный объект может быть удален и нажатием клавиш Delete и Backspace. Одиночный объект может быть удален этими клавишами без выделения.

Копирование объекта/объектов

Выделить объект, нажать Ctrl-X/Ctrl-C, переместить курсор в нужное место и вывести в него буфер, используя клавиши Ctrl-V. Таким образом объект может быть размножен многократно.

29.1. Генератор отчетов

Приступить к созданию программы, генерирующей новый отчет, можно через главное меню системы (пункты FILE+NEW+REPORT). При этом справа от строки основного меню появляется пункт Report, через который можно получить доступ непосредственно к меню генератора отчетов — REPORT-меню. Для работы с генератором требуется открытая база данных. Она может быть открыта предварительно через меню открытия файлов (пункты FILE+OPEN+DATABASE) либо командой USE <файл БД>. Если отчет формируется из нескольких баз данных, они все должны быть открыты в различных рабочих областях. Это можно сделать и непосредственно из генератора.

При работе с генератором создается два типа файлов (с расширением имени FRX и FRT), которые хранят образ отчета и с помощью которых выполняется его вывод. Здесь же может сохраняться информация о состоянии среды в момент создания образа отчета (имена открытых баз данных, используемые индексы и т.д.). Последнее необходимо, скорее, пользователю, работающему непосредственно в среде СУБД, но может быть полезно и программисту. Обычно к проектированию отчета приходится возвращаться многократно. В этом случае сохранение параметров среды освобождает от необходимости каждый раз открывать нужные файлы.

Другой способ вызвать генератор — это использовать команду создания файла-генератора отчета

■ CREATE REPORT [<FRX-файл>]

либо команду модификации, если такой <файл> уже существует:

■ MODIFY REPORT [<FRX-файл>]

Расширение имени не обязательно.

Если воспользоваться системным меню для вызова генератора или не задать

имя FRX-файла в команде CREATE, то FoxPro сам даст файлу имя UNTITLED.FRX ("Безымянный"). По завершении работы с генератором система предложит вам заменить это имя на более подходящее.

При переходе в генератор отчетов на экране появится специальное окно формирования вида отчета, в котором находится условное отображение будущего отчета — планшет.

Основное меню генератора отчетов с перечислением его функций изображено на рис.29.1.

Report		
Page Layout...		- вызов диалога форматирования страницы
Page Preview... ^I		- предварительный просмотр проекта отчета
Data Grouping...		- вызов диалога группирования данных отчета
Title/Summary...		- вызов диалога назначения заголовка/подвала
Variables...		- вызов диалога создания переменных
Box	^B	- создание линий/рамоч
Field...	^F	- вызов диалога построения выражений
Text	^T	- ввод текста в отчет
Add Line	^N	- добавление строки планшета
Remove Line	^O	- удаление строки планшета
Bring to Front	^G	- прямой порядок обхода выбранных полей
Send to Back	^J	- обратный порядок обхода полей
Center		- центрирование объекта на странице
Group		- выбранное множество объектов объявляется группой; в дальнейшем при выделении любого из элементов группы выделяется вся группа
Ungroup		- отмена группирования
Quick Report...		- вызов меню генератора "быстрого отчета"

Рис.29.1

Более детальное разъяснение смысла пунктов меню будет дано на конкретном примере создания отчета для базы данных KADR.DBF. Здесь упомянем только пункты меню Bring to Front и Send to Back. Они определяют, в каком порядке будут обрабатываться выражения, включаемые в отчет. Это может иметь значение, только когда одно выражение является функцией других.

Положим, база данных KADR.DBF уже открыта, а также командой CREATE REPORT дано имя файлу отчета KADR.FRX. Приступить к проектированию отчета удобно с пункта Quick Report главного меню генератора (REPORT-меню), который откроет экран диалога (рис.29.2).

Quick Report:

(.) Column Layout

() Form Layout

[X] Titles

[] Fields...

[] Add Alias

<< OK >>

Field1 Field2

xxxxxx xxxxxx

xxxxxx xxxxxx

< Cancel >

Рис.29.2

Элементы экрана означают следующее:

Column/Form Layout — отчет будет выводиться колонками/строками. Во внутреннем прямоугольнике окна диалога условно отображена выбранная форма (здесь вывод колонками).

Titles — заголовками колонок отчета будут названия полей.

Fields — открытие меню выбора полей, участвующих в отчете, из базы данных. По умолчанию в отчет выбираются все поля.

Add Alias — названия полей будут составными, включая псевдоним (Alias) базы данных. Это необходимо в случае, если в отчет выводятся данные из нескольких баз.

Положим, нам нужны не все поля, а только некоторые (FAM, TAB, DTR, POL, DET, SZAR, PER), которые мы отберем, используя диалог Fields (рис.29.3).

Database Fields:						Selected Fields:	
DTR	D	8	0	< Move->		KADR.FAM	
POL	C	1	0	< All->		KADR.TAB	
SEM	C	1	0			KADR.DTR	
DET	N	1	0	< Remove >		KADR.POL	
PODR	C	15	0			KADR.DET	
SZAR	N	7	2	<Remove All>		KADR.SZAR	
PER	M	10	0			KADR.PER	
Database:				<< OK >>			
KADR				< Cancel >			

Рис.29.3

Здесь в окне Database Fields перечислены все поля БД вместе с их атрибутами (типом и размером), а в окне Selected Fields — отобранные поля. Для отбора полей в отчет следует нажать клавишу Enter или дважды кнопку мыши на нужном поле. Можно сначала пометить (применяя также клавишу Shift) нужные поля, а затем их отобрать в правое окно, используя кнопку <Move>. Удаление некоторых или всех полей из отобранного множества осуществляется с помощью кнопки <Remove>/<Remove ALL>. Отбор всех полей — выбором кнопки <All>.

Внизу в POPUP-меню Database указано имя открытой базы (KADR.DBF). Теперь, если вернуться назад, мы увидим уже заполненный планшет отчета (рис.29.4).

KADR.FRX							
R:	5	C:	4	Move	Page Footer		
PgHead	Fam		Tab	Dtr		Pol	Det
PgHead						Szar	Per
PgHead							
PgHead							
Detail	fam		tab	dtr		p	d
PgFoot						szar	per
PgFoot	-						
PgFoot							
PgFoot	DATE()						
PgFoot						Page	_PAG

Рис.29.4

В верхней строке планшета (PgHead) отображены заголовки колонок, а в центре — сами выводимые поля (строка Detail), окрашенные контрастным цветом, и в них для удобства пользователя еще раз указаны их названия. Однако, если длины поля не хватает, здесь отображается только уместившаяся часть имени поля или выводимого выражения (P вместо POL, D вместо DTR, _PAG вместо _PAGNO).

Ширина отчета определяется шириной планшета, который ограничивается справа вертикальной чертой из символов '■', а снизу — сплошной полосой синего цвета. По умолчанию ширина отчета 80 колонок. Если все поля в них

не укладываются, уменьшается выводимая часть символьных полей. Выводимые имена полей обрезаются до фактической ширины этих полей. Положение границ легко может быть изменено с помощью мыши. Однако правая граница не может быть перемещена на место какого-то существующего объекта. Для этого предварительно нужно передвинуть сам объект.

Кроме того, сам планшет находится в системном окне, положением и размещением которого также можно управлять. Последнее важно в случае, если отчет (обычное явление) шире размера экрана.

Самая верхняя строка (статус-строка) планшета включает указание в нем на координаты курсора (на рис.29.4 — строка 5, колонка 4), режим курсора (Move) и положение курсора в области планшета (Page Footer). Отсчет строк/столбцов ведется с нуля.

Курсор имеет следующие режимы: Move, в котором он может свободно перемещаться на экране (действует по умолчанию), режим ввода текста Text, режим работы с полями базы Field, режим работы с рамками Box. В режим Text можно перейти, нажав любую содержательную клавишу, Ctrl-T, либо выбрав пункт Text в REPORT-меню. В этом режиме можно вводить любой текст на экране и он воспроизведется в отчете.

Аналогично в режимы Box и Field можно перейти с помощью меню или нажав клавиши Ctrl-B и Ctrl-F.

Возврат в режим Move осуществляется нажатием клавиши Enter или Escape. В последнем случае изменения, сделанные в объекте, не сохраняются.

Планшет может иметь следующие области, которые указаны (сокращенно) надписями в левой части экрана:

Title	— заголовок всего отчета;
Page Header	— заголовок каждой страницы отчета;
Detail	— содержание отчета;
Page Footer	— подвал каждой страницы;
Summary	— подвал всего отчета;
Group Header/Footer	— группирование данных при выдаче.

Строки могут быть легко дополнены с помощью мыши. Для этого нужно выделить ее название и "отбуксировать" вниз/вверх. Удаляемые строки, в которых есть данные, должны быть сначала очищены, или же должен быть использован пункт REPORT-меню Remove Line (клавиши Ctrl-O). Дополнение строк также может быть сделано через пункт Add Line (Ctrl-N).

Положение курсора в одной из областей отражается в статус-строке. Надпись Group Header/Footer — только в статус-строке.

По виду планшета мы видим, что отчет не имеет общего заголовка, подвала и группировки данных. Заголовок страницы будет просто состоять из имен полей, содержимое — из содержимого отобранных полей, а в подвале страницы будут выводиться текущая дата (DATE()), слово Page (страница) и номер страницы, которую определяет значение системной переменной PAGENO.

Контрольный просмотр отчета может быть выполнен нажатием клавиш Ctrl-I или выбором пункта Page Preview в REPORT-меню. Его фрагмент приведен на рис.29.5.

Горизонтальная штриховая линия обозначает место перехода на новую страницу, а вертикальная — правую границу отчета.

При работе с генератором, в особенности пока нет никакого опыта, целесообразно почаще прибегать к просмотру формируемого отчета.

Здесь даты (дата рождения и системная дата) отображены в американском формате. Впоследствии в программе даты будут выводиться в формате,

указанном командой SET DATE TO. Чтобы перейти к привычному виду даты, можно, не покидая генератора отчетов, вызвать командное окно по Ctrl-F2 и ввести команду SET DATE TO GERMAN.

Fam	Tab	Dtr	Pol	Det	Szar	Per
СИДОРОВ П.С.	13	10/12/56	М	1	350.00	Рабочий це
02/08/92						Page 1

Рис.29.5

Рассмотренный отчет, конечно, слишком примитивен. Усложним его. При этом выполним и группировку данных, например по полю DET (количество детей). Группировка позволит выводить записи группами в которых собраны записи, где родители имеют одинаковое количество детей (0, 1, 2, ...).

Фрагмент готового отчета, который мы создаем, приведен на рис.29.6.

С П И С О К							сегодня: 08.02.92
NN	Фамилия/Перемещ:	Таб	Год/р	Пол	Детей	С/зар	Премия
0	1	Стр. 2,	1 3	4	5	6	7
1	ПОПОВ А.А.	Количество детей - 0					
		234	1946	Мужчина	0	500.00	100
0	1	Стр. 2	4 3	4	5	6	7
9	ПОТАПОВ Д.П.	Количество детей - 3					
	Перемещения:	С 02.07.85	98 1960	Мужчина	3	280.00	56
		89 - начальник смены в ОГМ - мастер цеха окраски, с 06.12.					
Средняя зарплата		331.11	Фонд заработной платы			2980	
Инспектор по кадрам:							

Рис.29.6

Здесь в заголовке всего отчета кроме шапки отчета выводится текущая дата со словом "сегодня".

В заголовке каждой страницы — нумерация колонок и номер страницы. Вывод базы сгруппирован по количеству детей — сначала идут все работники у которых нет детей, затем те, у кого один ребенок и т.д. Вывод полей FAM, TAB, SZAR, DET изменений (кроме нового положения) не претерпел. Другие данные выводятся иначе. Так из поля DTR (Дата рождения) выводится только год рождения (функция YEAR(DTR)). В зависимости от значения поля POL (Пол), выводятся слова "Мужчина/Женщина" (функция IIF(POL='М', 'Мужчина', 'Женщина')), мемо-поле PER (Перемещения) теперь попало во второй ряд отчета и выводится несколькими строками фиксированной длины (по умолчанию мемо-поле выводится строкой длиной в десять символов). Кроме того, выводится новая величина — Премия, которая будет составлять 20% от средней зарплаты, но не более 1000 рублей и без учета копеек (функция INT(MIN(1000, 0.2 * SZAR))). Все работники теперь последовательно нумеруются (колонок NN). В подвале всего отчета вычисляются средняя зарплата по всей базе и фонд заработной платы (суммарная зарплата), округленный до рублей.

Планшет генератора, позволивший построить приведенный отчет, показан на рис.29.7. Его можно формировать из исходного планшета в любом порядке, но сначала давайте обратимся к пункту Page Layout в REPORT-меню, который раскроет следующее окно диалога (рис.29.8).

KADR.FRX									
R: 10 C: 2 Move Page Footer									
Title	С П И С О К								сегодня:DATE()
Title	NN	Фамилия/Перемещ:	Таб	Год/р	Пол	Детей	С/зар	Премия	
PgHead	Стр.		PAG						
PgHead	0	1	2	3	4	5	6	7	
PgHead									
1-det	Количество детей - d								
Detail	nom	fam	tab	year	iif(pol	d	szar	int(
Detail	Перемещения: per								
1-det									
PgFoot									
Summary	Средняя зарплата szar Фонд заработной платы round(s								
Summary									
Summary	Инспектор по кадрам:								

Рис.29.7

Page Layout:		< Options... >	
Page length	(rows):	66	
Top margin	(rows):	0	
Bottom margin	(rows):	0	<< OK >>
Printer indent	(columns):	0	
Right margin column:		80	< Cancel >
Environment:			
<input type="button" value="Save"/> <input type="button" value="Restore"/> <input type="button" value="Clear"/>			
[] Printer Driver Setup...			

Рис.29.8

Здесь содержатся важные параметры форматирования отчета: количество строк на странице (Page length), отступ сверху и снизу (Top margin и Bottom margin), число колонок левого отступа (Printer indent) и ширина всего отчета (Right margin column). Параметр Printer indent действует только при выдаче на принтер и в файл. Параметр Right margin column определяет предельную ширину отчета и отражает фактическое положение правой границы планшета. Введя новое число в эту колонку, установим новую ширину отчета, но лучше это сделать непосредственно в планшете, передвинув границу с помощью мыши.

Меню Environment предназначено для сохранения, восстановления, очистки параметров среды в отчете. Если здесь доступна только кнопка <Save>, среда не запоминается. При выборе этой кнопки активируются и остальные. Если здесь выбрать <Restore>, среда сохранится, если <Clear> — информация о ней (если ранее была) удаляется.

Один самый первый раз при выходе из генератора FoxPro сам запрашивает пользователя о необходимости сохранения операционного окружения фразой "Save environment information?".

Кнопка <Options...> выводит в окно диалога, изображенное на рис.29.9.

Его пункты имеют (сверху вниз) такой смысл:

- о прогон страницы перед печатью отчета;

- о прогон после печати;
- о не выводятся заголовки страницы (Page header);
- о в отчет выводятся только заголовки и итоги, данные (Detail) пропускаются;
- о подавление вывода пустых строк из базы данных;
- о к именам при создании отчета полей добавляется псевдоним базы данных (применяется при работе с несколькими базами).

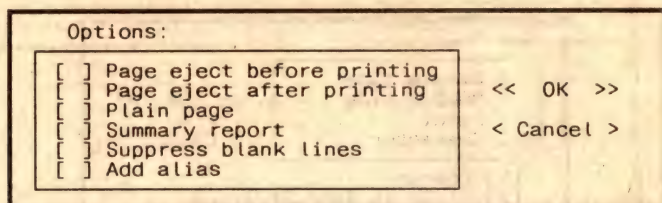


Рис.29.9

Снова вернемся к планшету. Создадим в нем области заголовка (Title) и подвала (Summary) отчета. Для этого обратимся к пункту Title/Summary в REPORT-меню, который выведет нас на следующий экран (рис.29.10).

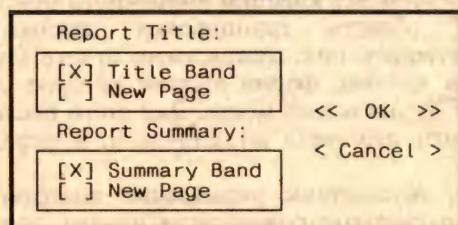


Рис.29.10

Установка знака [X] в кнопках Title Band и Summary Band вызовет появление в планшете по одной строке типа Title и Summary (заголовок и подвал всего отчета). Выбор кнопок New Page повлечет выделение для этих элементов отчета отдельных страниц.

Далее с помощью клавиатуры или мыши количество таких строк может быть легко увеличено до нужного числа. В нашем случае — до трех и четырех соответственно.

Приступим непосредственно к формированию отчета.

Заполним заголовок отчета. В нулевую строку отчета внесем его название "СПИСОК". Сделать это проще всего, нажав букву С в нужном месте, а по окончании ввода нажать клавишу Enter. Аналогичным образом по ходу работы будем вводить другие текстовые строки.

Объект DATE() перенесем со старого места в правый верхний угол и снабдим его словом "сегодня". В первой строке отчета изобразим двойную горизонтальную линию. Для этого установим курсор в ее будущее начало (координаты 1,0) и нажмем Ctrl-B либо выберем пункт Box в меню генератора. Появится маленький прямоугольник, которому мы можем с помощью мыши или клавиш придать любые размеры и положение. Такая линия будет одинарной. Чтобы сделать ее двойной, нужно выделить созданную линию. При этом возникнет окно диалога (рис.29.11).

Пользуясь им, можно выбрать вид линии: одинарная (Single), двойная (Double), сплошная полоса (Panel) или линия заполненная любым символом,

который мы можем выбрать из меню, развертываемого к пункту Character.

Кнопка Comment здесь и далее будет открывать экран для ввода комментариев к данному объекту (чтобы не забыть, для чего он) и на вывод данных в отчет никак не влияет. Использование ее не имеет большого смысла.

Сначала изменим заголовки полей. Это можно сделать, введя новые заголовки (уже по-русски) поверх старых либо написав их все в новой строке, а старую уничтожив целиком.

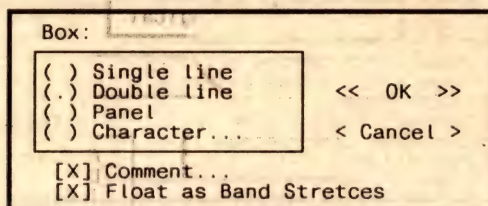


Рис.29.11

Далее заполним заголовок страницы (строки PgHead). В первую его строку сначала поместим переменную _PAGENO (номер страницы) и слово "Стр.". С обеих сторон номера страниц уже известным способом проводим двойные линии. Затем пронумеруем все колонки выводимого отчета (0, 1, 2, ...).

Пропустив пока область группировки данных (1-det), перейдем непосредственно к формированию содержимого отчета (строк Detail).

Сначала перенесем данные, форма выдачи которых не изменяется (FAM, TAB, POL, SZAR, DET), на новые места. Для этого последовательно выделим, как указывалось выше, эти поля на планшете и передвинем их на новые места.

Теперь, например, осуществим управление выводом даты рождения, из которой печатается только год рождения. Сначала перенесем поле DTR на новое место. Затем выделим это поле и обратимся к пункту Field в REPORT-меню, или нажмем клавишу Enter, или дважды кнопку мыши, или Ctrl-F. В ответ будет предъявлено окно (см. рис.29.12).

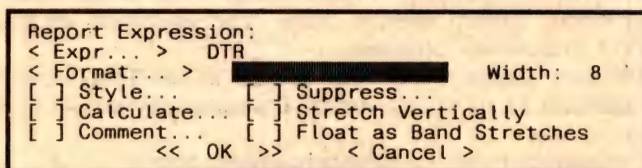


Рис.29.12

Здесь в пространстве, отведенном для задания выражения указано поле DTR. Теперь можно сразу ввести выражение YEAR(DTR) либо, если вы не помните вид функции или она достаточно сложна, можно перейти в окно построителя выражений, выбрав кнопку <Expr...> (рис.29.13).

Здесь в области ввода Report <expr> формируется желаемое выражение. Сейчас это просто имя поля DTR. В верхней части экрана изображены четыре POPUP-меню, через которые доступны все функции FoxPro, разбитые на группы, — математические, строковые, логические функции и функции работы с датами (Math, String, Logical, Date). Если вы не помните, как именно выглядит нужная функция, можно обратиться в соответствующее меню и поискать ее там. Нажатие клавиши Enter повлечет немедленное перенесение ее в строку формируемого выражения. В нашем случае,

поскольку аргумент уже есть, нужно сначала поставить курсор под первую букву имени поля DTR, а затем обратиться к меню Date и выбрать там функцию "YEAR(expD)", после чего отредактировать строку нужным образом (YEAR(DTR)). При возврате в планшет имеет также смысл уменьшить длину выводимого выражения с восьмью (ранее было установлено для поля DTR) до четырех позиций (Width: 4).

Рис.29.13

Построитель выражений позволяет создать любое сколь угодно сложное выражение, используя как различные поля базы, так и переменные, которые можно взять через меню Field Names и Variables. Здесь же через меню Database можно обратиться и к любой ранее открытой в другой рабочей области базе данных. Тогда в окне Field Names появится список полей из этой базы. В окне Variables сейчас представлены только системные переменные, которые могут быть полезными при формировании отчета. Однако мы можем назначить и собственные переменные. Построенное таким образом выражение целесообразно проверить на синтаксическую правильность, используя кнопку <Verify>. Если выдано сообщение "Expression is valid", значит, ошибок нет и выражение можно использовать в отчете.

Рис.29.14

Теперь рассмотрим остальные пункты диалога Report Expression: **Format** — задание формата выдачи данных. Здесь (рис.29.14) можно непосредственно ввести символы задания шаблона и/или функций вывода либо вызвать диалог их выбора. В нашем случае для переменной типа даты, каким является поле DTR, будет предложено следующее меню, в котором можно задать один из двух типов установок вывода даты — Edit "SET" Date (формат, зависящий от текущей команды SET DATE TO) и British Date (европейский формат даты). При работе с датами удобно использовать первую установку. Она действует и по умолчанию. Другие типы полей генерируют другое содержимое меню Editing Options, соответствующее шаблонам и функциям ввода-вывода для команды @ ... SAY/GET.

Width — видимая длина выводимых данных из этого поля в отчете. Для

выражения YEAR(DTR) его можно уменьшить до четырех. Это же можно сделать с помощью мыши прямо в планшете.

Style — управление типом шрифта.

Suppress — подавление повторяющихся данных. Если рядом в базе находятся записи, имеющие одинаковое значение поля, оно выводится только один первый раз. В следующих строках оно замещается пробелами. По умолчанию подавления не происходит (Off). Такой режим целесообразен в упорядоченных базах. На рис.29.15 приведено окно диалога для поля DTR.

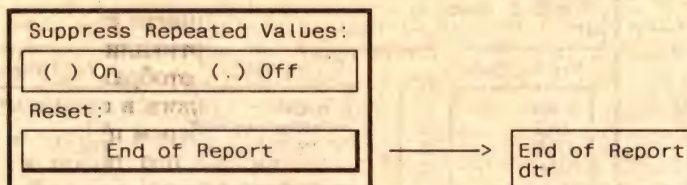


Рис.29.15

В POPUP-меню Reset можно указать границы действия этой установки: до конца отчета, до конца страницы (со следующей страницы действие установки возобновляется и повторяющееся поле выводится один раз).

Calculate — назначение возможного типа вычислений "по вертикали".

По умолчанию вычисления не выполняются (Nothing). Возможно вычисление (рис.29.16) следующих величин: подсчет числа записей (Count), суммирование значения поля (Sum), нахождение среднего арифметического (Average), минимального и максимального значений (Lowest/Highest), среднеквадратического отклонения (Std. Deviation) и дисперсии (Variance).

С помощью кнопки Reset устанавливается диапазон, в котором происходит расчет. Если это End of Report, то сквозной до конца отчета, если End of Page, то на каждой странице он начинается сначала.

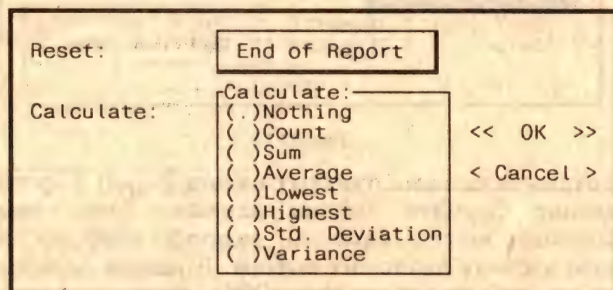


Рис.29.16

Stretch Vertically — установление режима выдачи, когда поле выводится в несколько строк одинаковой длины (указанной для него на планшете) до достижения его конца. Удобно использовать с мемо-полями или символьными полями значительной длины, например с полем PER.

Float as Band Stretches — позволяет управлять положением полей, расположенных на планшете ниже тех, для которых выбрана установка Stretch Vertically. Если этого не сделать, то произойдет наложение

данных. Пункт Float as Band Stretches указывает генератору отчетов, что указанное поле должно располагаться не в фиксированной строке, а ниже, т.е. только после завершения вывода предыдущего поля с заранее неизвестным количеством строк. В нашем случае не предполагается вывод каких-либо данных ниже поля PER.

Как в предыдущем случае, построим выражение для колонки ПОЛ (IF(POL='M','Мужчина','Женщина')) и колонки ПРЕМИЯ (INT(MIN(1000, 0.2*SZAR))). Здесь следует также изменить длину отображения данных. Так, вместо длины поля POL один символ теперь нужно задать семь, а для поля Премия — четыре.

В колонке NN (номер по порядку) в планшете создадим переменную NOM, для которой назначим вычисление по вертикали количества записей. Для этого установим курсор на желаемое место отображения будущей переменной NOM, нажмем клавиши Ctrl-F и в окне диалога в строке <Expr...> введем имя NOM, а с помощью кнопки [] Calculate выберем пункт () Count.

Поле PER расположим непосредственно под полем FAM и зададим ему режим выдачи с разбивкой на строки — Stretch Vertically, а также увеличим размер по горизонтали до правой границы отчета.

В подвале всего отчета должны находиться сводные характеристики по зарплате — Средняя зарплата и Фонд заработной платы. Для этого здесь создаем два новых выражения, в которых используются режимы вычислений Average и Sum соответственно. Причем во втором случае, поскольку по условию требуется выдача суммы, округленной до рублей, используется выражение вида ROUND(szar,0).

Для группировки базы по полю DET она должна быть отсортирована или проиндексирована по нему. С этой целью через командное окно, вызванное с помощью клавиш Ctrl-F2, не покидая генератор, создадим индексный файл KADRDET.IDX, используя команду

```
INDEX ON det TO kadrDET COMPACT
```

или откроем его, если он уже есть.

Далее обратимся к пункту REPORT-меню Data Grouping, из которого мы попадем в пустое пока окно задания группируемых переменных (рис.29.17). Здесь мы можем удалить и изменить выражение (<Delete> и <Change>) либо добавить новое (<Add>). Сейчас нам доступно только последнее. Выбор кнопки <Add> выведет нас в экран, изображенный на рис.29.18.

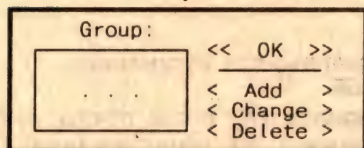


Рис.29.17

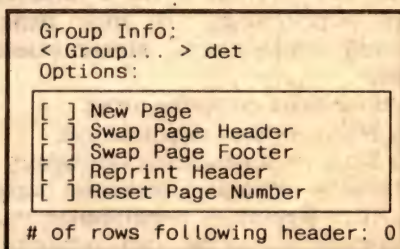


Рис.29.18

В области ввода, указанной непосредственно после кнопки <Group...>, можно задать групповое выражение (здесь — DET) или в более сложном случае обратиться к построителю выражений.

Остальные опции экрана перечислены ниже.

New Page — каждая группа начинается с новой страницы.

Swap Page Header — заголовок группы (Group Headers) выводится вместо

заголовка страницы (Page Headers) на каждой новой странице.

Swap Page Footer — то же, но подвал группы (Group Footers) замещает подвал страницы (Page Footers).

Reprint Header — заголовок группы (Group header) выводится на каждой странице.

Reset Page Number — нумерация страниц начинается заново при изменении значения выражения с групповой переменной.

Rows Following Group Header — указывает число пустых строк, добавляемых в подвал группы при выводе.

Теперь после возврата в планшет мы увидим две новые строки, содержащие в левой части слова "1-det". Верхняя из этих строк соответствует строке заголовка группы (Group Footers), нижняя — строке подвала группы (Group Header). В дальнейшем к меню группировки можно попасть уже не только через главное REPORT-меню, а нажав дважды кнопку мыши на этих словах.

Таким образом, используя левую колонку планшета, можно обращаться к диалогам, формирующим соответствующие области планшета. Такие элементы колонки окрашены в синий цвет.

Далее в заголовок группы для наглядности внесем текст "Количество детей" и поле DET. Из-за того, что это поле имеет длину в один разряд на планшете, мы увидим только одну букву d.

В отчете можно использовать не только поля базы данных, но переменные, которые могут являться сложными функциями от полей, других переменных и пользовательскими функциями. Их применение позволяет достигнуть большей гибкости при формировании отчета. Создание переменных осуществляется через пункт Variables в REPORT-меню.

Сначала мы попадаем в окно Variables, где указаны все имеющиеся к данному моменту переменные отчета (рис.29.19).

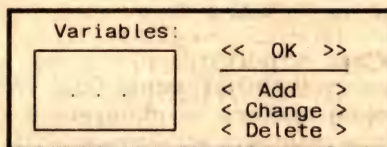


Рис.29.19

Сейчас оно пустое, и мы с помощью кнопки <Add> обратимся к экрану создания переменных. В этом окне (рис.29.20) мы можем задать имя переменной, определить вычисляемое ею выражение и задать исходное значение.

Элементы окна описаны ниже.

Variable Name — имя переменной.

Value to Store — выражение, которому приравнивается переменная.

Initial Value — исходное значение переменной.

Release After Report — переменная уничтожается [X] после печати отчета. В противном случае она остается в памяти до использования команд CLEAR ALL/MEMORY.

Reset — задается граница, по которой переменная сбрасывается в исходное значение. По умолчанию — конец отчета.

В качестве примера здесь создана переменная A, в которой подсчитывается число работников, имеющих зарплату более 2000 руб ($A = \text{IIF}(\text{SZAR} > 2000, A + 1, A)$). Исходное значение переменной A равно 0.

После создания переменной ее можно поместить в планшет. Для этого, нужно перейти в режим FIELDS (клавишами Ctrl-F или через REPORT-

меню) и указать ее имя в строке ввода или взять из списка переменных в окне Variables построителя выражений. Однако в данном случае это не имеет большого смысла. Будем считать, что переменная A нужна уже после завершения вывода отчета. Поскольку установка Release After Report оставлена пустой ([]), значит, переменная не будет уничтожена и останется доступной для анализа со статусом PUBLIC.

Variable Name: A	Calculate:
<Value to Store...>	() Nothing
IIF(SZAR>2000,A+1,A)	() Count
<Initial Value...>	() Sum
0	() Average
[] Release After Report	() Lowest
Reset: <input type="text" value="End of Report"/>	() Highest
	() Std. Deviation
	() Variance

Рис.29.20

В отчете допускается широкое использование ПФ. Для этого необходимо, нажав клавиши Ctrl-F, вызвать меню задания полей и, не обращаясь к построителю выражений, прямо ввести имя ПФ, например FFF(). При этом, конечно, должна существовать соответствующая функция с именем FFF.

Завершение сеанса работы с генератором и выход из планшета осуществляются привычным способом с помощью клавиш Ctrl-W или через центральное меню (пункты FILE+SAVE). Если вы хотите выйти без сохранения сделанных изменений, нужно нажать клавишу Escape.

Для выдачи отчета в программе следует использовать команду

```
■ REPORT [FORMAT <FRX-файл>] [<границы>]
  [FOR <вырL1>]      [WHILE <вырL2>]
  [HEADING <вырC>]   [NOEJECT]
  [NOCONSOLE]        [NOOPTIMIZE]
  [PLAIN]             [PREVIEW]
  [TO PRINTER/TO FILE <файл>]
  [SUMMARY]           [PDSETUP]
```

Опции команды:

FORMAT <FRX-файл> — задает имя FRX-файла, используемого для построения отчета.

HEADING <вырC> — указывает выражение, выводимое в дополнительной строке заголовка отчета на каждой ее странице.

NOEJECT — подавляется прогон листа, предшествующий печати отчета.

NOCONSOLE — подавляет параллельный вывод отчета на экран при выдаче его на принтер или в файл.

PLAIN — подавляется выдача заголовков страницы.

PREVIEW — выдача отчета для его предварительного просмотра в специальное окно, идентичное окну, вызываемому при нажатии клавиш Ctrl-I в генераторе отчетов.

TO PRINTER/TO FILE <TXT-файл> — выдача отчета на принтер или в текстовый файл.

SUMMARY — выдаются только итоговые строки отчета. Сами данные игнорируются.

PDSETUP — используется драйвер принтера, определенный и сохраненный

при формировании формы отчета в генераторе отчетов.

Пример. Выдача отчета, сгенерированного с помощью форматного файла KADR.FRX, в текстовый файл SPRAVKA.TXT. Выдача на консоль подавляется.

```
REPORT FORMAT kadr TO spravka NOCONSOLE
```

Отчет формируется из данных базы KADR.DBF. В общем случае ее нужно предварительно открыть (вместе с индексом KADRDET.IDX), если при выходе из генератора не было сохранено состояние среды системы. Если же это было сделано, то база будет открыта автоматически. Если она уже и так открыта, открытие будет осуществлено повторно в свободной рабочей области.

Хороший способ изучения генератора отчетов (и других генераторов) — это разбор готовых FRX-файлов, которых много в пакете FoxPro. Их розыск может быть предпринят через главное меню (пункты FILE+OPEN+REPORT).

К сожалению, никакого аналога текстового программного файла (подобного PRG-файлу) генератор не создает. Однако можно воспользоваться внешней утилитой FRX2PRG.EXE фирмы Platinum Software International, которая преобразует FRX-файл в PRG-файл.

29.2. Генератор экранов

При разработке экранов создаются следующие типы файлов.

Собственно генератор экранов формирует два типа файлов с расширениями SCX и SCT, в которых хранится образ экрана на планшете, и состояние среды (открытые базы, индексы).

Для построения экранной программы требуется прибегнуть к генератору приложений, который создает SPR-файл, являющийся текстовым файлом программы. При этом должна быть доступна утилита FoxPro GENSCRN.PRG. Затем такая программа может быть запущена на исполнение командой DO <SPR-файл>. При этом она компилируется в SPX-файл.

Screen		Общие опции генератора
Screen Layout...		- форматирование экрана
Open All		- предьявление всех процедур, связанных с данным экраном
Snippets	~S	Средства создания объектов экрана
Box	~B	- изображение линий/прямоугольников
Field...	~F	- создание/редактирование полей на экране
Text	~T	- создание/редактирование текстов
Push Button...	~H	- создание триггерных кнопок
Radio Button...	~N	- создание радиокнопок
Check Box...	~K	- создание кнопок-переключателей
Popup...	~O	- создание скрытых POPUP-меню
List...	~L	- создание списков меню
Inv. Button...	~I	- создание невидимых кнопок
Bring to Front	~G	Эти пункты доступны при выделении объекта
Send to Back	~J	- прямой порядок обхода выбранных полей
Center		- обратный порядок обхода полей
Reorder Fields		- центрирование поля на экране/окне
Color...		- перенумерация по порядку выделенных полей
Group (Boxes)		- окраска выделенных объектов
Ungroup		- группирование выделенных объектов
Quick Screen...		- снятие группирования
		FC - диалог быстрого формирования экрана

Рис.29.21

Обращение к генератору экранов для создания нового или редактирования старого экрана может быть осуществлено через главное меню (пункты FILE+NEW/OPEN+SCREEN). При этом новый экран получает название

UNTITLE. Другой способ — использование команды

■ CREATE/MODIFY SCREEN <SCX-файл>

Вид SCREEN-меню и краткое разъяснение его пунктов приведены на рис.29.21.

Создадим экран ввода-редактирования в базу KADR.DBF. Для этого лучше всего обратиться к пункту Quick Screen, в котором нам будет предложено открыть базу данных (если она еще не открыта), и через его меню (рис.29.22) определить исходный вид экрана

```

Quick Screen:
( ) By Column Layout
(.) By Row Layout
[X] Titles
[ ] Fields...
[X] Add Alias
[ ] Memory Variables
Maximum field width: 80
<< OK >>      < Cancel >

Field1 xxxxxx
Field2 xxxxxx
Field3 xxxxxx
  
```

Рис.29.22

Здесь сначала предлагается указать способ размещения полей на экране — одной/несколькими колонками (By Column/Row Layout).

Поскольку сейчас создается всего лишь предварительный экран, то это все равно. Остановимся на способе By Row Layout. Следующие три позиции меню аналогичны таким же в генераторе отчетов. Кнопка Memory Variables определит создание экрана не для полей БД, а для одноименных переменных, но с префиксом "m" (например, m.fam). Это предусмотрено при построении экранных форм для временных переменных с копированием их из/в БД. Параметр Maximum field width определяет предельную длину поля. Если наша экранная форма создается на экране терминала, то его размер не может превышать 80 символов в строке. Если ранее было определено специальное окно, то длина определяется шириной этого окна.

С помощью кнопки Field обратимся к уже знакомому по генератору отчетов диалогу отбора полей на экран и возьмем оттуда поля FAM, TAB, DTR, POL, PODR, SZAR и PER. После возврата на уровень планшета мы увидим экран, приведенный на рис.29.23.

KADR.SCX		
R:	C:	Field
1	1	fam.....
2	2	
3	3	dtr..
4	4	
5	5	podr.....
6	6	szar
7	7	per.....

Рис.29.23

Как и раньше, верхняя строка планшета — это статус-строка. В самом планшете в левом столбце перечислены имена отобранных полей, а рядом справа — выделенные контрастным цветом области ввода. В них указываются номера полей в порядке, с которым они отбирались, еще раз имена полей (если они умещаются в установленной для них области ввода) и многоточие до окончания поля.

Теперь мы можем перемещать поля, устанавливать для них другую длину, делать новые надписи и создавать даже некоторые средства обработки. При определении длины здесь имеется в виду длина отображаемой на экране части, а не фактическая длина поля в базе, которая изменена быть не может.

Окончательный вид планшета показан на рис.29.24.

KADR.SCX

R 5 C: 12 | Move |

Личные данные

Сегодня 1: date(

Табель 2: 04 Фамилия 3: fam.....

Дата рождения 4: dtr..... Ср. зарплата 5: szar

Подразделение 6: podr..... Пол (М/Ж) 7

Перемещения 8: rep.....

(Выход - Tab)

или ^End/W)

<Назад> <Вперед> <Начало> <Конец> <Удаление> <Выход>

Просмотр документов

Рис.29.24

Здесь окантовка двойной линией с заголовком "Личные данные" принадлежит специально определенному нами окну, которое мы рассмотрим позже. Все данные на планшете соответствуют их будущему отображению на экране. Обсудим его заполнение.

Во-первых, полям соответствуют заголовки, написанные по-русски, и изменены их местоположение и порядок следования. Кроме того, видим, что изменились и номера полей — они расположены в возрастающем порядке. Это важно, поскольку в соответствии именно с этими номерами, а не с фактическим расположением на экране определяется очередность доступа к полям. Если бы мы оставили номера такими, как они были в исходном планшете, то курсор перемещался бы от поля FAM к полю TAB, а не наоборот. Этот порядок через SCREEN-меню адаптируется под желаемый для выделенных полей (пункт Reorder Fields).

Field:

(.) Say () Get () Edit

< Say... > date() << OK >>

< Format... > Range: < Cancel >

[] Upper... [] Lower...

[] When... [] Error... [] Scroll bar

[] Valid... [] Comment... [] Allow tabs

[] Message... [] Disabled [] Refresh

Рис.29.25

Вообще говоря, не обязательно, чтобы порядок доступа к полям определялся их расположением на экране. Порядок может быть любым. Перенумерацией можно управлять и через пункты меню — Bring to Front и Send to Back. Так, если вы вручную поочередно выделите поля в желаемом порядке и нажмете Ctrl-G (т.е. выберите Bring to Front), то все поля перенумеруются в порядке их обхода в момент выделения. При нажатии Ctrl-J порядок изменится на обратный. Следует отметить, что в этом смысле к полям относятся и кнопки.

Теперь рассмотрим конкретные поля. В левом верхнем углу выводится

текущая дата. Для этого через пункт FIELD в SCREEN-меню вызывается диалог определения облика поля (рис.29.25).

Здесь в верхней строке мы выбираем кнопку ()SAY, а после другой кнопки <SAY...> вводим функцию DATE(), т.е. определяем команду вида @...SAY DATE(). В более сложных случаях, возможно, имеет смысл обратиться к построителю выражений, который отсюда может быть вызван. Кроме того, здесь можно указать формат вывода (<Format...>), верхний и нижний диапазоны переменной (Upper/Lower), а также большинство опций команды (см. описание команд @...SAY/GET/EDIT).

Следующие поля (TAB, FAM, DTR, SZAR и PODR) не имеют каких-либо особенностей. В поле POL (Пол) разрешается ввод только двух значений М и Ж. Этот вопрос удобно решить с помощью шаблона "@М М,Ж". Данные, введенные в диалоге FIELD для поля POL, изображены ниже:

```
< Get... > kadr.pol
< Format... > @М М,Ж
```

Теперь выбор нужного значения может осуществляться с помощью клавиши Space. Чтобы пользователь об этом знал, через кнопку Message обратимся к следующему окну (рис.29.26).

Message:

()Procedure <Edit...> (.)Expression	<< OK >> < Cancel >
'Нажмите клавишу Пробел'	

Рис.29.26

Поскольку здесь требуется создать только сообщение, выберем кнопку Expression и введем в окне редактирования предложение 'Нажмите клавишу Пробел'.

Последнее поле, на которое нужно обратить внимание, — это мемо-поле PER (Перемещение). Для него удобно использовать команду @...EDIT (выбрать кнопку <EDIT> в FIELD-диалоге), а затем довести на планшете это поле до необходимых размеров. Теперь содержимое мемо-поля будет предьявляться в трех строках экрана без нажатия клавиш Ctrl-Home.

Кроме собственно ввода/редактирования данных на экране можно реализовать средства управления с помощью кнопок. Для этого воспользуемся триггерными кнопками (Push Button). Выбрав соответствующий пункт REPORT-меню, мы перейдем в диалог создания такого кнопочного меню. Пусть нам нужно реализовать пять операций: переход Назад/Вперед на одну запись, переход в Начало/Конец базы, Удаление/Восстановление записи и Выход из экрана.

Эти действия могут быть спроектированы через окно, приведенное на рис.29.27.

В его левой части укажем надписи-приглашения, которые будут размещены в кнопках. Справа мы установим, что кнопки должны быть расположены горизонтально (Horizontal). Через Spacing можно задать расстояния между нашими кнопками (по умолчанию — один символ). Через Terminating можно указать, является ли наше меню завершающим команду READ. В разделе Variable следует задать переменную, которая воспримет выбор из меню. Дадим ей имя K.

В разделе Options определим реакции на выбор из меню. Через кнопку

Valid обратится к экрану, приведенному на рис.29.28.

Здесь, выбрав кнопку Procedure, запишем саму процедуру (DO CASE ...). Если здесь слишком тесно, можно обратиться непосредственно к редактору через кнопку Edit. Сейчас мы видим только небольшую часть процедуры. Полностью она будет приведена позже.

Рис.29.27

Рис.29.28

Кроме того, здесь же мы организуем Message-сообщение о действиях пользователя ('Выберите нужный режим') и номере текущей записи. Через кнопку Message зададим следующее выражение (Expression):

Выберите нужный режим Текущая запись - '+str(recno(),3)

Теперь обратимся к диалогу форматирования всего созданного экрана. Для этого выберем пункт Screen Layout в SCREEN-меню, который откроет нужный экран диалога (рис.29.29).

Кнопка DeskTop устанавливает вывод спроектированного нами экрана непосредственно на экран терминала, а кнопка Window — в заданное окно. Выберем окно. Тогда нам будет предложено задать его имя, заголовок и нижнюю надпись (Name, Title, Footer). В разделе Size устанавливаются высота и ширина окна (11x68). Эти параметры соответствуют размеру планшета. Если вас перестал устраивать размер окна, нужно переместить на планшете его границы, вернуться в Screen Layout-окно, выбрать кнопку DeskTop, а затем снова кнопку Window. При этом описание окна получит новые размеры, а остальные параметры сохранятся старыми. Однако никакое окно не может иметь размеры, меньшие, чем необходимо для размещения всех его элементов, даже если вы уменьшили планшет.

В разделе Position можно задать произвольные координаты левого верхнего угла окна. По умолчанию оно центрируется (Center) на экране.

В разделе Screen Code мы можем задать команды, которые будут вставлены в программу перед и после предъявления сгенерированного экрана (Setup и

Cleanup & Procs). Вызвав диалог Setup, можно поместить команды открытия файлов, команды вида SET, осуществить очистку экрана и другие подготовительные действия. В диалоге Cleanup наоборот — действия по завершению ввода-редактирования.

() Desktop
Name: kadr
Title: Личные данные
Footer: Просмотр документов
Size: Screen Code:
Size:
Height: 11
Width: 68

(.) Window
<Type...>
[] Setup...
[] Cleanup & Procs...

Position: READ Clauses:
Row: [] Activate... [] Show...
Column: [] Valid... [] When...
[X] Center [] Deactivate...

Environment: [X] Add alias
< Save > < Restore > < Clear >

<< OK >>
< Cancel >

Рис.29.29

В разделе Environment предусмотрена возможность сохранения в SCX-файле состояния среды системы.

В разделе READ Clauses можно предусмотреть обработку опций команды READ. Обычно к ним прибегают при многооконном экране.

Через кнопку Type мы попадем в окно диалога установления параметров окна (рис.29.30).

Type: User

Attributes: [] Close [] Float [X] Shadow [] Minimize

Border: () None (.) Single () Double () Panel () System

Color Schemes:
Primary: User Winds
Popup: User Menus

<< OK >>
< Cancel >

Рис.29.30.

В верхнем POPUP-меню Type можно выбрать вид окна одного из системных типов — System, Dialog, Alert или установить собственный тип — User. В последнем случае для нас становятся доступны нижеперечисленные средства его настройки.

В разделе Attributes задаются операции управления окном — закрытие, перемещение, наличие тени, стягивание в одну строку.

В разделе Border указывается тип окантовки — отсутствие окантовки, одинарная, двойная линия, сплошная полоса или подобно системным окнам — с символами управления окном.

В разделе Color Schemes устанавливаются цвета (цветовые схемы) для самого окна (Primary) и связанных с ним POPUP-меню.

Можно считать, что создание планшета экрана завершено. Если выйти, нажав клавиши Ctrl-W, будут созданы SCX- и SCT-файлы. Но чтобы получить "живой" экран, необходимо, не выходя из генератора экранов, обратиться через пункт главного меню Program к строке Generate, которая выведет нас в окно диалога (рис.29.31).

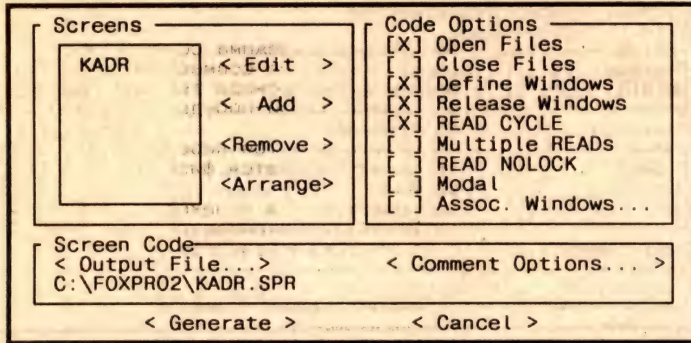


Рис.29.31

В левом внутреннем окошке Screens изображен список экранов (у нас только KADR), которые можно редактировать, добавлять, удалять и располагать в произвольном порядке (кнопки Edit, Add, Remove, Arrange). Эти возможности важны при конструировании многооконного интерфейса.

В разделе Code Options указываются опции, сопровождающие генерацию экрана. Вот основные.

Open Files — автоматическое открытие файлов баз данных, индексов.

Close Files — закрытие файлов после завершения работы с окном.

Define Windows — по команде DEFINE WINDOW определяются окна, описанные в диалоге Screen Layout. В противном случае они должны быть определены в программе предварительно.

Release Windows — по завершении работы с окном оно уничтожается командой RELEASE WINDOWS.

READ CYCLE — команда READ используется с опцией CYCLE.

Multiple READs — каждый экран ввода снабжается своей командой READ.

Modal — добавляется опция MODAL.

Associated Windows — указываются доступные внешние для нашей команды READ окна, например окно калькулятора. Этот пункт добавляет опции MODAL и WITH <окна> к команде READ.

В разделе Screen Code задается имя программного SPR-файла (KADR.SPR) и в диалоге <Comment Options...> — некоторая дополнительная информация (имя автора программы, форма комментариев, директория, куда помещается файл др.).

После завершения всех установок инициируется генерация программы (пункт Generate). Далее следует освободить экран от планшета и через командное окно ввести команду (расширение имени файла обязательно)

```
DO kadr.spr
```

которая и предъявит спроектированный экран на терминале.

Программный файл KADR.SPR может быть взят в редактор для просмотра и изменения. Он может использоваться в качестве процедуры совместно с

другими программами.

С некоторыми сокращениями текст файла KADR.SPR приведен ниже.

```
*-----Программа формирования экрана KADR.SPR-----
#REGION 0
REGIONAL m.currarea, m.talkstat, m.compstat
IF SET("TALK") = "ON"      && Выяснение режима SET TALK
    SET TALK OFF          && и запоминание его исходного состояния
    m.talkstat = "ON"
ELSE
    m.talkstat = "OFF"
ENDIF
m.compstat=SET("COMPATIBLE") && Выяснение режима совместимости
SET COMPATIBLE FOXPLUS     && Установление совместимости с FoxBASE+
m.currarea=SELECT()        && Выяснение номера текущей рабочей области
IF USED("kadr")            && Если в какой-нибудь области открыта база
    && KADR.DBF,
    SELECT kadr            && осуществляется переход в эту область и
    SET ORDER TO 0         && устанавливается физический порядок записей
ELSE
    SELECT 0               && Иначе
    && производится переход в свободную область,
    && ищется и открывается база KADR.DBF
USE (LOCFILE("kadr.dbf", "DBF", "Where is kadr?"));
    AGAIN ALIAS kadr ORDER 0
ENDIF
*
*   Window definitions   определение окна
*
IF NOT WEXIST("kadr")      && Если окно KADR не было ранее
    && определено, оно определяется
    DEFINE WINDOW kadr FROM INT((SROW()-11)/2), INT((SCOL()-67)/2);
    TO INT((SROW()-11)/2)+10, INT((SCOL()-67)/2)+66 ;
    TITLE "Личные данные" FOOTER "Просмотр документов" ;
    NOFLOAT NOCLOSE SHADOW DOUBLE COLOR SCHEME 10
ENDIF
*
*   KADR Screen Layout   формирование экрана
*
#REGION 1
IF WVISIBLE("kadr")      && Если окно KADR активно,
    ACTIVATE WINDOW kadr SAME && оно выводится на передний план
ELSE
    ACTIVATE WINDOW kadr NOSHOW && Иначе
    && оно активируется, но пока не
    && предьявляется на экране
ENDIF
&& Определение содержимого окна
@ 1,6 SAY "Табель"
@ 1,25 SAY "Фамилия"
@ 2,6 SAY "Дата рождения"
@ 3,48 SAY "Пол"
@ 4,6 SAY "Перемещения"
@ 3,6 SAY "Подразделение"
@ 0,39 SAY "Сегодня"
@ 7,0 TO 7,64
@ 2,38 SAY "Ср. зарплата"
@ 0,51 SAY DATE() SIZE 1,8
@ 1,15 GET kadr.tab SIZE 1,3 DEFAULT 0
@ 1,34 GET kadr.fam SIZE 1,25 DEFAULT " "
@ 2,20 GET kadr.dtr SIZE 1,8 DEFAULT { / / }
@ 2,52 GET kadr.szar SIZE 1,7 DEFAULT 0
@ 3,20 GET kadr.podr SIZE 1,15 DEFAULT " "
@ 3,58 GET kadr.pol SIZE 1,1 DEFAULT " "
    PICTURE "@M М,Ж" MESSAGE 'Нажмите клавишу Пробел'
@ 4,20 EDIT kadr.per SIZE 3,39,0 DEFAULT " "
    && Описание кнопок управления базой
@ 8,0 GET k ;
    PICTURE "@*HN Назад;Вперед;Начало;Конец;Удаление;Выход" ;
    SIZE 1,10,1 DEFAULT 1 VALID q3a0rg84i()
    MESSAGE 'Выберите нужный режим Текущая запись - '+';
    STR(RECNO(),3)
@ 6,7 SAY "или End/W)"
@ 3,52 SAY "(М/Ж)"
@ 5,6 SAY "(Выход Tab"
IF NOT WVISIBLE("kadr")
    ACTIVATE WINDOW kadr
ENDIF
```



```

READ CYCLE                                && Команда чтения данных
RELEASE WINDOW kadr                        && Удаление окна KADR
*
*      Closing Databases      закрытие базы
*
* Восстановление среды, существовавшей вначале
IF USED("kadr")                          && Закрывается вновь открытая база KADR
  SELECT kadr                            && (если была открыта данной программой)
  USE
ENDIF
SELECT (m.currarea)                      && Возврат в исходную рабочую область
#REGION 0
IF m.talkstat = "ON"                     && Восстановление исходного параметра
  SET TALK ON                           && команды SET TALK
ENDIF
IF m.compstat = "ON"                     && Восстановление исходного параметра
  SET COMPATIBLE ON                     && команды SET COMPATIBLE
ENDIF

*-----функция _Q3AORG84I обработки кнопок, в которую передается-----
*-----переменная k из условия VALID для Push Button-----
*
*      Q3AORG84I      k VALID
*      Function Origin:
*      From Screen:    KADR,      Record Number: 20
*      Variable:       k
*      Called By:       VALID Clause
*      Object Type:     Push Button
*      Snippet Number: 1
*
FUNCTION _q3a0rg84i                      && k VALID
#REGION 1
DO CASE
  CASE k=1                                && <Назад>
    SKIP -1
    IF BOF()
      GO TOP
    ENDIF
  CASE k=2                                && <Вперед>
    SKIP
    IF EOF()
      GO BOTTOM
    ENDIF
  CASE k=3                                && <Начало>
    GO TOP
  CASE k=4                                && <Конец>
    GO BOTTOM
  CASE k=5
    IF DELETE()                          && <Удаление>
      RECALL
    ELSE
      DELETE
    ENDIF
  CASE k=6                                && <Выход>
    CLEAR READ
ENDCASE
@ 0,3 SAY IIF(DELETE(), 'Удалено', ' ')
SHOW GETS
RETURN

```

Структура программы очевидна. В самом ее начале осуществляются подготовительные действия. Выполняется команда SET TALK OFF, открываются база данных KADR.DBF и окно KADR. Для совместимости с прежними версиями пакета вводится команда SET COMPATIBLE FOXPLUS.

Как видим, генератор экранов сам дает имена процедурам. Так, единственная у нас процедура обработки кнопок получила имя _Q3AORG84I.

Некоторые разъяснения дадим к обработке кнопки "Удаление". Здесь при выборе этой кнопки (k=5) происходит анализ функции DELETE(). Если ее значение .F. (т.е. запись не имеет пометки об удалении), то она помечается. Если наоборот — пометка снимается. Одновременно об этом выводится сообщение в нулевой строке окна. По завершении программы исходная среда

восстанавливается.

Некоторые строки, очевидно, можно изъять в окончательном варианте программы.

В тексте программы встречается новая команда, вернее, директива компилятору

■ #REGION <номер региона>

При создании экрана пользователю иногда приходится вводить и свои переменные. При этом возможно, что некоторые из них получают одинаковые имена. Для того чтобы избежать конфликта между такими переменными, процедура автоматически разбивается на регионы с помощью рассматриваемых директив. Параметр <номер региона> может быть в диапазоне от 0 до 31.

После этого командой

■ REGIONAL <список переменных/массивов>

можно объявить внутренние переменные для этого региона. Такие переменные имеют все свойства локальных переменных типа PRIVATE, но не внутри процедуры, а внутри региона. Если при компиляции процедуры встречается идентичное имя другой такой переменной, оно получает новое уникальное имя путем добавления к его оригинальному имени символов подчеркивания и номера региона.

29.3. Генератор меню

Генератор позволяет легко создавать иерархическое меню для прикладных систем. Строится оно на базе главного системного меню СУБД (_MSYSMENU) и получает все его свойства.

При формировании меню могут быть созданы четыре типа файлов:

MNX и MNT

— запоминают образ создаваемого меню;

MPR

— текстовый программный файл, полученный после генерации;

MP

— откомпилированный программный меню-файл.

Для формирования программы, реализующей построенное меню (т.е. MPR-файла), необходимо присутствие в текущей директории файла GENMENU.PRG.

Приступить к созданию/редактированию меню можно через главное меню (пункты FILE+NEW/OPEN+MENU) или с помощью команды

■ CREATE/MODIFY MENU <MNX-файл>

KADR.MNX			
Prompt	Result	Options	
Система	Submenu	< Edit >	[X]
База	Submenu	< Edit >	[X]
Выход	Proc.	< Edit >	[X]

Menu Bar

< Try it >

Item

< Insert >

< Delete >

Рис.29.32

При создании меню возникает пустой планшет (рис.29.32). Его наполнение пока игнорируем. Рабочая часть планшета имеет четыре столбца. В столбец Prompt заносятся пункты-приглашения создаваемых меню. При этом для

первого уровня меню генерируется команда вида

```
DEFINE PAD <имя PAD-пункта> OF _MSYSMENU PROMPT <приглашение>
```

В столбце Result указываются действия, которые вызовет выбор каждого из пунктов. Здесь возможно указание одного из следующих объектов:

Command — задание некоторой <команды>, которая будет выполняться при выборе данного пункта меню. Если это пункт самого верхнего уровня меню, в меню-программу добавляется команда вида

```
ON SELECTION PAD <имя PAD-пункта> OF _MSYSMENU <команда> ,
```

либо (для строк POPUP-меню следующих уровней) команда

```
ON SELECTION BAR <номер пункта меню> OF <имя POPUP-меню> <команда>
```

Submenu — другое вспомогательное POPUP-меню. В программу для горизонтального меню вводится команда

```
ON PAD <имя PAD-пункта> OF _MSYSMENU ACTIVATE POPUP <имя POPUP-меню>
```

или (для пункта POPUP-меню) команда

```
ON BAR <номер пункта> OF <имя POPUP-меню>;  
    ACTIVATE POPUP <имя другого POPUP-меню>
```

Pad Name — здесь можно задать собственное, а не генерируемое автоматически имя Pad-пункта в команде вида

```
DEFINE PAD <имя PAD-пункта> OF _MSYSMENU PROMPT <приглашение>
```

Здесь подразумевается <имя> одного из PAD-пунктов системного меню, которое вы хотите использовать в конструируемом меню. Каждый элемент системного меню имеет свое специальное имя, которое можно выяснить, например, воспользовавшись пунктом Quick Menu в MENU-меню или используя функцию SYS(2013).

Bar # — задание в качестве строки создаваемого POPUP-меню элемента одного из системных POPUP-меню.

Proc. — процедура. За текущим пунктом меню закрепляется некоторая процедура.

Для горизонтального меню к программе добавляется команда

```
ON SELECTION PAD <имя PAD-пункта> OF _MSYSMENU;  
    DO <имя процедуры> IN <имя программы, содержащей процедуру>
```

Для пункта POPUP-меню

```
ON SELECTION BAR <номер пункта> OF <имя POPUP-меню>;  
    DO <имя процедуры> IN <имя программы, содержащей процедуру>
```

Здесь <имя процедуры> — уникальное имя, назначаемое генератором меню. Сама процедура может быть написана непосредственно тут же. В зависимости от того, какой объект задан, в этом же столбце откроется доступ к их созданию/редактированию через кнопку Create/Edit, либо, если выбран пункт Command — сделать непосредственный ввод команды.

Options	
Comment:	<input type="text"/> << OK >>
< Cancel >	
<input type="checkbox"/> Shortcut...	<input type="checkbox"/> Mark...
<input type="checkbox"/> Skip For...	<input checked="" type="checkbox"/> Pad Name...

Рис.29.33

Столбец Options открывает доступ к окну, изображенному на рис.29.33. Здесь можно установить следующие опции для элементов меню:

- Shortcut — назначение KEY-клавиш вызова данного пункта меню;
 Mark — указание символов пометки выбранных пунктов;
 Skip For — условие, исключающее доступ к данному пункту;
 Pad Name/Bar # — собственное имя заголовка, номер строки меню.

При разработке меню удобнее задавать собственные (а не использовать предлагаемые генератором случайные, например Q3615TTMV), понятные программисту имена Pad-пунктов (Pad Name) и BAR-меню. Сейчас здесь можно задать Pad Name.

В правом верхнем углу планшета отображено скрытое POPUP-меню, где показано имя текущего уровня меню. В исходном состоянии это всегда Menu Bar, т.е. главное меню. В дальнейшем через POPUP-меню можно перемещаться вверх по уровням создаваемого меню. Каждый новый уровень автоматически получает системное имя, которое далее может быть заменено на другое. Вниз по уровням можно перемещаться непосредственно через колонку Result в рабочей части планшета.

Остальные кнопки имеют следующий смысл:

- Try it — просмотр внешнего вида меню.
 Insert — вставка нового пункта меню.
 Delete — удаление выбранного пункта меню.

Кроме планшета в системном меню появляется пункт Menu, через который можно получить доступ к меню генератора (рис.29.34).

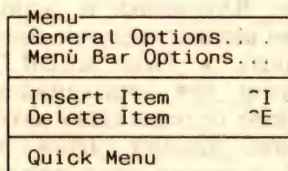


Рис.29.34

Пункты меню:

General Options — открывает окно диалога, приведенное на рис.29.35.

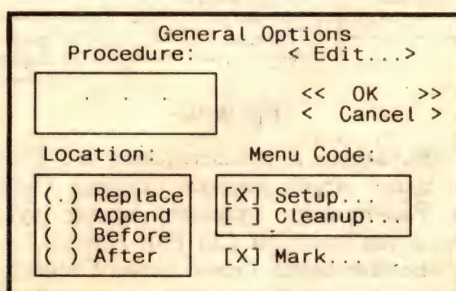


Рис.29.35

Здесь кнопки Replace и Append в разделе Location устанавливают создаваемое меню вместо или дополнительно справа к системному меню. Кнопки Before и After — перед или после выбранного пункта системного меню. В прикладных системах, скорее, имеет смысл режим Replace.

В разделе окна Menu Code можно ввести команды предваряющие/завершающие (Setup/Cleanup) установку собственно меню.

Кнопка Mark дает возможность установить вид символа пометки выбранных пунктов меню.

В окне Procedure (или через кнопку Edit) можно ввести <команду>, которая будет включена в команду следующего вида:

ON SELECTION MENU _MSYSMENU <команда>

Menu Bar Options — вызывает окно диалога, подобное окну General Options, в котором мы можем установить свои имена всем вложенным меню, задать команду, которая при генерации меню будет подставлена в команду вида

ON SELECTION POPUP ALL <команда>

Insert/Delete — вставка/удаление пунктов меню.

Quick Menu — для редактирования предьявляется системное меню, которое мы можем приспособить для своих целей. Мы можем даже русифицировать его (но только два верхних уровня — строку главного BAR-меню и все POPUP-меню). Привязка элементов системного меню к его пунктам может быть выяснена в HELP FoxPro (пункт MENU — System Menu Names) или в документации.

На рис.29.36 схематически изображено меню, которое мы построим с помощью генератора для работы с базой KADR.DBF. Оно состоит из строки главного трехкомпонентного BAR-меню и дополнительных POPUP-меню. Пункту "Система" подчинено вспомогательное меню, позволяющее выполнять упаковку и удаление данных в базе KADR.DBF. К опасному пункту "Удаление" вызывается меню для подтверждения выбора пользователя. Элемент "База" главного меню осуществляет вызов вспомогательного меню, где доступны редактирование данных через готовую экранную форму, просмотр и печать отчета, а также последовательный поиск записей по фамилии и табельному номеру.

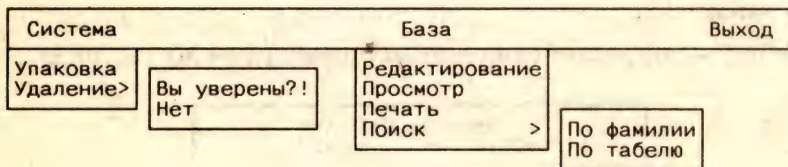


Рис.29.36

Теперь приступим собственно к проектированию. Самый первый планшет был изображен выше. Здесь через кнопки столбца Options в диалоге дадим свои имена каждому Pad-пункту главного меню: system, baza, vih. Эта возможность предоставляется кнопкой [X] Pad Name...

Через кнопки Edit (вначале здесь стоит слово Create) создадим следующие приведенные ниже планшеты.

Для того чтобы проследить цепь совершенных действий, в левом верхнем углу каждого планшета нами указаны пункты меню, для которых они созданы. В POPUP-меню планшета (в его правом верхнем углу) автоматически формируются имена вложенных меню, которые, как уже говорилось, целесообразно (но необязательно) заменить на свои. Сделать это можно через пункт Menu Bar Options, который откроет окно диалога с пунктом Name, где мы можем ввести собственное имя. В нашем случае по мере создания меню будем задавать следующие имена: syst, udalen, baza (совпадает с именем Pad-пункта), poisk.

Следующий планшет (рис.29.37) формирует POPUP-меню к пункту "Система".

Система	Command pack	[]	syst
Упаковка	Submenu < Edit >	[]	
Удаление			

Рис.29.37.

Для упаковки используется только одна команда PACK, а для удаления создается новое вспомогательное (Submenu) меню (рис.29.38).

Система+Удаление			udalen
Вы уверены?!	Command zap	[]	
Нет	Command	[]	

Рис.29.38

Здесь выбору "Нет" не соответствует никакая команда.

В POPUP-меню "База" для редактирования базы по команде DO KADR.SPX загружается созданный нами ранее экран. Пункты "Просмотр" и "Печать" выполняются с помощью также сгенерированного ранее формата отчета (команды REPORT FORM KADR PREWIEV и REPORT FORM KADR TO PRINT NOCONSOLE). На экране (рис.29.39) видны только начальные фрагменты команд. Остальные их части могут быть доступны при перемещении курсора.

База			baza
Редактирование	Command do kadr.spk	[]	
Просмотр	Command report form kad	[]	
Печать	Command report form kad	[]	
Поиск	Submenu < Edit >	[]	

Рис.29.39

"Поиск" реализуется через вспомогательное меню (рис.29.40), где в обоих пунктах выполняется одна и та же процедура POISK.PRG.

Поиск			poisk
По фамилии	Command do poisk	[]	
По таблицю	Command do poisk	[]	

Рис.29.40

Пункт главного меню "Выход" реализуется через внутреннюю (вводимую прямо в окне генератора и без имени) процедуру

```
RELEASE WINDOW poisk
CLOSE ALL
```

В MENU-меню через пункт General Options и кнопку Setup вводим команды предварительных установок, включая открытие базы и описание окна POISK для задания в нем поискового ключа:

```
CLEAR
DEFINE WINDOW poisk FROM 4,25 TO 6,70
USE kadr
```

Оба эти фрагмента после генерации будут слиты с основной программой. Кроме того, здесь вызывается одна внешняя процедура POISK.PRG для поиска в базе:

```
ACTIVATE WINDOW poisk
n=RECNO()
```



```

DEFINE POPUP poisk MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF poisk PROMPT "По фамилии"
DEFINE BAR 2 OF poisk PROMPT "По таблице"
ON SELECTION BAR 1 OF poisk DO poisk
ON SELECTION BAR 2 OF poisk DO poisk

```

```

*
*   Q3C0PDH24 ON SELECTION PAD
*   Procedure Origin:
*   From Menu:      KADR.MPR           Record:   19
*   Called By:     ON SELECTION PAD
*   Prompt:        Выход
*   Snippet:       1
*

```

```

PROCEDURE q3c0pdh24 && Процедура выхода
RELEASE WINDOW poisk
CLOSE ALL

```

Как видим, созданная система меню не имеет команды описания главного меню (команды DEFINE MENU), а в командах DEFINE PAD в позиции имени горизонтального меню употребляется имя главного меню FoxPro — MSYSMENU. Кроме того, наше меню не имеет команды активации. Фактически мы построили подобие системного меню, которое постоянно доступно и может быть вызвано в состоянии ожидания из любого места программы, а также из окна Command клавишей F10/Alt или мышью (или любой назначенной клавишей).

Однако такая программа-меню сама по себе не может быть главной программой прикладной системы, поскольку после ее завершения не создается состояние ожидания, как при обращении к обычному меню, и программа завершается. Разработчики FoxPro рекомендуют для создания состояния ожидания применять команду READ после определения меню. При этом, используя ее опции, можно организовывать гибкое управление обработкой данных, в том числе интегрируя вместе меню и окна ввода.

При желании легко превратить такое меню в обычное. Для этого нужно в начало программы внести строку-описатель главного BAR-меню, где задать собственное произвольное имя BAR-меню:

```
DEFINE MENU <имя BAR-меню>...
```

а затем всюду заменить слово MSYSMENU на выбранное вами <имя BAR-меню>. Сделать это можно, используя режим поиска и массовую замену в редакторе FoxPro. Теперь активация такого меню, как обычно, будет возможна по команде ACTIVATE MENU <имя BAR-меню>, которую следует разместить ниже самой последней команды описания меню.

Формируемый с помощью генератора программный код состоит из четырех частей.

1. Setup Code — код настройки. Здесь инициализируются нужные переменные, открываются базы, описываются окна.

2. Menu Definition — код определения меню. Определяется конфигурация меню и его реакции.

3. Cleanup Code — код очистки. Сюда включаются команды, идущие после завершения определения меню. В полученной программе этого раздела нет.

4. Procedures — процедуры. В конце сгенерированного кода меню приписываются все внутренние процедуры. Здесь — процедура _Q3C0PDH24, где удаляются все окна и закрываются файлы.

Возврат к системному меню может быть осуществлен вводом команды SET SYSMENU TO DEFAULT в окне Command.

Готовая меню-программа может быть загружена по команде

```
DO <MPR/MPX-файл>
```


в зависимости от того, было (MPX) или еще нет (MPR) выполнено компилирование. Например, в данном случае — по команде DO KADR.MPR.

В заключение отметим, что программные коды, создаваемые как генератором меню, так и генератором экранов, с одной стороны несколько избыточны, а с другой — используют не все возможности языка. Вместе с тем такие программы-полуфабрикаты легко адаптировать и развивать для собственных целей, что делает их полезным инструментом разработчика. Вообще, хотя оба эти генератора допускают написание программистом собственных процедур не выходя из них, вряд ли кто станет пытаться строить законченные системы таким образом. Область их применения — скорее, построение именно меню и экранов, которые в дальнейшем, возможно, "улучшаются" и включаются в создаваемую прикладную систему.

29.4. Менеджер проектов

Менеджер проектов позволяет программисту легко следить за всеми программными (и другими) файлами, участвующими в работе создаваемой прикладной системы. По существу, это каталог файлов, из которого можно легко извлекать все его компоненты и модифицировать их. Кроме того, здесь можно слить все модули системы в один, а также сформировать исполнимые программные файлы. Проект можно использовать уже в самом начале разработки системы, сохраняя его окно на экране. Используя генератор проектов, можно легко слить все FRX-программы в один APP-файл.

Работа менеджера проектов похожа на действие утилиты Make, имеющейся во многих компиляторах алгоритмических языков. При работе с проектом производится постоянное сравнение дат создания исходных и объектных модулей. Проект работает только с самыми последними вариантами программ. Все данные о проекте сохраняются в специальных файлах проектов с расширениями PJX и PJT.

Приступить к созданию/редактированию проекта можно через главное меню (пункты FILE+NEW/OPEN+PROJECT) или по команде

■ CREATE/MODIFY PROJECT <PJX-файл>

На рис.29.41 изображен уже заполненный планшет менеджера проектов. Рабочая часть планшета состоит из двух столбцов. Столбец Name содержит имена файлов, а столбец Type — типы (Отчет, Меню, Экран, Программа и т.д.). Головной файл помечен точкой.

KADR. PJX		
Name	Type	
KADR	Report	< Edit >
KADR	• Menu	< Info >
KADR	Screen Set	< Add >
POISK	Program	< Remove >
		< Build >

Рис.29.41

В правой части планшета имеются кнопки управления:

- Edit — открытие любого файла проекта для модификации;
- Info — информация о составе проекта;
- Add — дополнение проекта файлами;
- Remove — удаление файлов из проекта;
- Build — обновление/создание проекта.

С помощью кнопки Build открывается окно диалога (рис.29.42) по созданию/обновлению проекта и формированию исполнимых программных файлов.

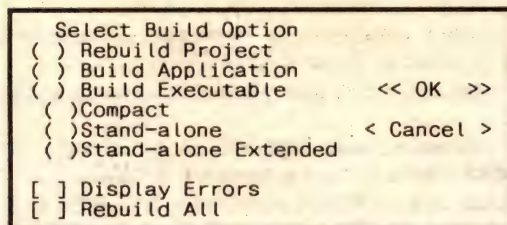


Рис.29.42

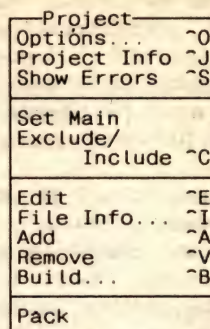


Рис.29.43

Кнопки экрана:

- Rebuild Project** — создание/обновление проекта.
- Build Application** — создание Приложения (программы в форме APP-файла). APP-файл объединяет в себе все компоненты проекта, образуя один программный модуль.
- Build Executable** — доступ к средствам создания EXE-файлов следующих видов в зависимости от дальнейшего выбора:
- Compact** — создание компактного EXE-файла. Такой файл, хотя и имеет расширение исполнимого файла, таковым не является. Компактная EXE-программа работает только в присутствии файлов библиотек FoxPro или самой СУБД FoxPro. По существу, это форма исполнительной системы-интерпретатора (RunTime).
- Stand-alone** — создание автономного EXE-файла.
- Stand-alone Extended** — создание автономного EXE-файла для расширенной версии FoxPro. Автономные EXE-файлы любого типа могут быть созданы и работают только при наличии пакета Distribution Kit.
- Display Errors** — на экране предьявляются ошибки, найденные в процессе построения проекта, которые заносятся в специальный файл с расширением ERR.
- Rebuild All** — обновляются все файлы проекта независимо от дат создания исходного и откомпилированного модулей.
- Кроме перечисленных средств управления для работы с проектом может использоваться специальное PROJECT-меню (рис.29.43) со следующими пунктами (совпадающие с перечисленными ранее опущены):
- Options** — вызов диалога задания вспомогательной технической информации.
- Project Info** — количественные данные о составе проекта.
- Show Errors** — просмотр файла ошибок, найденных при создании проекта.
- Set Main** — объявление текущего файла головным файлом проекта.
- Exclude/Include** — исключение (Exclude) из процесса компиляции текущего файла, помещенного в проект. При этом файл помечается символом "э". Этот режим необходим для файлов, которые присоединяются к проекту на этапе выполнения. Например, мы можем включить в проект и файлы баз данных. После компиляции проекта они также войдут в окончательный модуль, но с атрибутом "только-чтение". Так, это может иметь смысл для файла конфигурации среды FoxPro — FOXUSER.DBF. Однако другие базы данных в своем большинстве не могут быть объединены с программой. Вместе с тем в проект удобно

поместить все эти файлы с пометкой "э". Это исключит их включение в программу при компиляции, но откроет легкий доступ программисту к самим данным в базах через кнопку <Edit> планшета. Снятие пометки "внешний" выполняется выбором пункта Include.

Pack — упаковка записей, помеченных для удаления из PJX-файла.

Рассмотрим последовательность создания проекта на примере соединения всех созданных нами ранее файлов по обработке базы KADR.DBF.

После вызова планшета менеджера проектов следует выполнить следующие действия.

1. Через кнопку <Add> разыскать головной файл проекта и поместить его на планшет. В нашем случае — это файл меню также с именем KADR.

2. Через кнопку <Build> вызвать меню построителя проектов и в нем выбрать кнопку Rebuild Project. После этого FoxPro найдет все программные (включая экраны, отчеты, меню) файлы, на которые есть ссылки, и представит их имена на планшете. Головной файл помечается точкой. Именно такой планшет отображен в начале раздела. Далее, если нужно, можно поместить в отчет и все другие файлы, например DBF, пометив их как внешние.

3. По завершении формирования перечня файлов проекта можно создать сводный программный файл с расширением APP, если выбрать в меню построителя проектов пункт Build Application.

Далее можно построить компактный и независимый EXE-файлы при наличии пакета Distribution Kit.

Использование интерактивных средств работы с проектами позволяет решать задачи интеграции модулей системы и получения готового программного продукта. Вместе с тем имеются и соответствующие команды:

■ BUILD PROJECT <PJX-файл> FROM <список файлов проекта>

С помощью команды BUILD PROJECT создается база данных проекта (PJX-файл) из имен файлов, участвующих в проекте (программных файлов, файлов экранов, меню, отчетов, библиотек).

■ BUILD APP <APP-файл> FROM <PJX-файл>

Команда создает APP-файл, используя файл-проект (PJX-файл). Для работы APP-программы вне FoxPro нужны файлы библиотеки поддержки, содержащиеся в пакете Distribution Kit.

■ BUILD EXE <EXE-файл>
FROM <PJX-файл> [STANDALONE] [EXTENDED]

Команда создает EXE-файл, используя файл-проект. По умолчанию создается компактный EXE-файл. Хотя он имеет расширение имени EXE, он не является автономным и для его работы нужны практически те же компоненты пакета Distribution Kit, что и для выполнения APP-файлов. При создании EXE-файла одноименный APP-файл (если есть) уничтожается, и наоборот.

В зависимости от используемых опций могут быть также созданы следующие разновидности исполнимых, автономных файлов:

STANDALONE — стандартная версия EXE-файла, для которой нужна Стандартная версия FoxPro. В каталоге FoxPro должны быть следующие файлы: FOXPRO.LIB, FOXPROS.LIB, FOXCLIBM.LIB, FOXMATHM.LIB, WLINK8.EXE.

EXTENDED — расширенная версия EXE-файла. Должны быть доступны файлы: FOXPROX.LIB, FOXPROSX.LIB, FOXCLIBR.LIB, FOXMATHR.LIB, FOXLDR.EXE, WLINK8.EXE.

Глава 30. СРЕДСТВА ОТЛАДКИ ПРОГРАММ

При отладке программных продуктов часто бывает необходимо детально проследить поведение программы и ее переменных в сомнительных местах, т.е. выполнить трассировку.

Для этого желательно:

- выдавать на экран исполняемую в данный момент команду, а также значения нужных переменных/функций;

- замедлять темп выполнения программы и располагать возможностью управлять им вручную;

- иметь возможность приостанавливать в любой момент исполнение программы с целью ее анализа и, может быть, вмешательства с клавиатуры.

FoxPro обладает удобным и гибким отладчиком программ. Эти средства "находятся" в меню Window главного системного меню FoxPro.

Пункты WINDOW+DEBUG и WINDOW+TRACE выводят на экран два специальных отладочных окна Debug и Trace, которые могут вызываться раздельно или вместе. В окне Trace выводятся команды исполняемой программы, а в окне Debug — значения указанных в нем переменных и выражений. Оба окна представлены на рис.30.1 и 30.2.

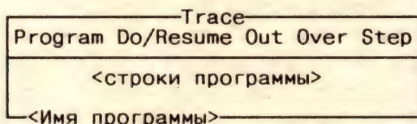


Рис.30.1

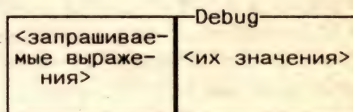
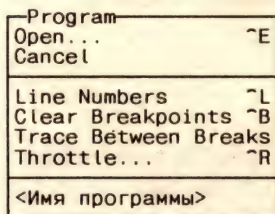


Рис.30.2

Находясь в окне Trace в момент останова программы, мы можем управлять трассировкой программы, пользуясь горизонтальным TRACE-меню, расположенным в верхней его части. Рассмотрим элементы этого меню.



диалог загрузки программных файлов
завершение приостановленной программы

строки программы нумеруются
удаление всех точек останова
выделение команд между точками
установка задержки выполнения команд
в диапазоне 0...5.5 с
имя исполняемой программы/процедуры

Рис.30.3

Program — управление выполнением и предъявлением программы. Этот пункт меню имеет собственное POPUP-меню (рис.30.3).

Open — аналогичен пункту DO всего TRACE-меню, но предоставляет "быстрые" клавиши Ctrl-E, которые, впрочем, действуют (как и все остальные — Ctrl-L, Ctrl-B, Ctrl-R) и вне меню только при наличии окна трассировщика.

Line Numbers — через этот пункт можно организовать вывод пронумерованных строк отлаживаемой программы. В дальнейшем, используя команду TYPE <имя командного файла> NUMBER, при желании можно вывести на экран или в файл и саму "пронумерованную" программу.

Trace Between Breaks — через этот пункт меню можно управлять трассировкой программы при наличии точек останова. По умолчанию все команды выводятся в окно Trace. Если выбрать

указанный пункт (слева появится ромбик), в окне будут фиксироваться только команды, на которых стоят точки останова. Обычно это удобнее, поскольку не приходится тратить время на просмотр смежных фрагментов программы, сразу оказавшись в нужном месте. Далее при желании мы можем перейти к пошаговому режиму.

Выбор последнего пункта (<Имя программы>) позволяет при необходимости быстро вернуться к команде, где произошла остановка, если вы переместились в другое место в окне Trase.

Do/Resume — в начальный момент предлагается (DO) меню открытия и запуска программных файлов. Программа вызывается и останавливается на первой команде для выбора режима просмотра и введения, если нужно, точек останова. Затем этот пункт меню заменяется на Resume — продолжение выполнения программы.

Out — быстрое завершение текущей программы/процедуры без ее дальнейшей трассировки (если далее нет точек останова).

Over — вызываемая в программе процедура не предьявляется в окне Trase (если в ней нет точек останова). Этот пункт меню действует в момент, когда текущей является команда вызова процедуры.

Step — пошаговый режим исполнения программы.

Если у вас есть сомнения, касающиеся не определенных мест программы, а поведения некоторых переменных, то следует начинать отладку с вызова окна Debug, в котором нужно указать эти переменные с установкой на них точек останова. В момент изменения этих переменных программа будет прервана, и можно будет вызвать окно Trase с программой, в которой при желании, в свою очередь, также можно указать любое количество точек останова.

Ниже (рис.30.4 и 30.5) приведены некоторые конкретные виды окон Debug и Trase с точками останова.

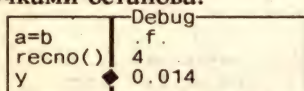


Рис.30.4

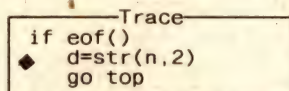


Рис.30.5

Здесь в окне Debug программистом запрашиваются значения трех выражений, из которых для переменной Y сделана пометка (на вертикальной разделительной линии) — точка останова. Это значит, что при изменении Y программа остановится. В левой части окна Debug программист имеет возможность указать до 16 переменных или выражений, значения которых будут отслеживаться в программе и индизироваться в правой части окна. Каждый ввод выражения/переменной в окно Debug должен обязательно сопровождаться нажатием клавиши Enter. Ввод точек останова удобно делать мышью. С клавиатуры это можно осуществить следующим образом:

- нажимается клавиша Tab,
- появившийся на вертикальной разделительной линии маркер перемещается в нужную строку,
- нажимается клавиша Space.

В окне Trase мы можем указывать точки останова и мышью, установив ее маркер в левую колонку, а также клавишей Space.

Снимаются точки останова аналогичным образом или с помощью пункта меню Clear Breakpoints.

Вообще при отладке исключительно удобно пользоваться мышью, поскольку обычно приходится часто перемещаться между окнами Trase, Debug, Command и, возможно, окнами, которые генерирует отлаживаемая

программа, а также для указания точек останова. Все точки останова сохраняются в программе до ее перекомпиляции или до их отмены.

Режим отладки и его средства могут быть вызваны и непосредственным применением следующих SET-команд, которые имеют только один параметр с двумя возможными значениями "Включено" и "Выключено" (OFF/ON). Ниже в синтаксисе команд параметр, значение которого принято по умолчанию, указан заглавными буквами.

- SET ECHO OFF/on — осуществляет выдачу всех исполняемых команд (параметр ON) программы в окно Trace.

- SET DEBUG off/ON — разрешает использование (ON) окон отладки. В режиме разработки программ параметр команды должен иметь значение ON. Последнюю компиляцию полностью готовой работы лучше выполнять при установке SET DEBUG OFF. Это немного уменьшит полученный объектный модуль и, главное, исключит возможность несанкционированного доступа к исходному тексту программы через окно Trace.

- SET TALK off/ON — организует автоматическую выдачу (ON) на экран результатов большинства исполняемых команд. Это средство малоудобно.

- SET STEP OFF/on — устанавливает пошаговый (ON) темп исполнения программы с вызовом окна Trace.

- SET STATUS OFF/on — выводит статус-строку на экран. Она может быть полезной при анализе программ, работающих с несколькими базами сразу.

Эти команды можно временно включить в текст программы в нужных местах с последующим удалением. Однако возможностей самого отладчика обычно совершенно достаточно.

При отладке программ, как правило, нет необходимости в пошаговом режиме просматривать ее всю. Это слишком медленно. Лучше ввести точки останова в подозрительные места программы. Для этого целесообразно осуществить, например, следующие действия.

- Вызвать пока пустое окно Trace.

- Через пункт меню DO в Trace-окне загрузить командный файл. При этом в окне Trace появится текст программы, по которому мы можем свободно перемещаться.

- При помощи курсора переместиться ниже в исследуемое место программы и расставить точки останова.

- Выбрать пункт Resume для продолжения выполнения программы до достижения точки останова.

- Далее можно, например, перейти в режим пошагового выполнения программы через пункт Step для изучения ее поведения, возможно, с привлечением окна Debug.

- После завершения сеанса отладки можно дать завершиться программе естественным образом (без трассировки), удалив окно Trace, либо прервать ее совсем, выбрав пункт меню окна Cancel.

Если на экране только окно Debug, содержащее точки останова, то программа будет выполняться в естественном темпе до тех пор, пока помеченная переменная/выражение не изменит своего значения. В этот момент выполнение программы приостановится, появится системное сообщение

Do suspended

и курсор переместится в командное окно. Это указывает на то, что СУБД выполнила команду

■ SUSPEND

В отличие от команды CANCEL данная команда не завершает выполнение программы, а только приостанавливает ее с возможностью возврата к выполнению.

Поскольку сейчас мы фактически находимся в программе, то можем в командном окне запросить значения любых выражений (командой ?), а также выполнить некоторые другие действия (например, сделать присвоения переменным или осуществить перемещения в базе), позволяющие провести полную диагностику программы.

По завершении отладочных действий мы можем вернуться в программу вводом в окне Command или через системное меню (пункты PROGRAMM+RESUME) команды продолжения

■ RESUME

можем вызвать и окно Trace и продолжить трассировку оттуда.

Окна отладчика доступны в любом из состояний ожидания, которые имеются в программе (обычно это команды ввода данных), поскольку при этом доступно системное меню FoxPro при нажатии клавиши Alt/F10.

Вообще при любом прерывании как вследствие неполадок в программе, так и от клавиши Escape возникнет меню-сообщение вида

```

*** INTERRUPTED ***
< Cancel >    < Suspend >    < Ignore >

```

через пункт <Suspend> которого можно временно выйти из программы как в командное окно, так и в системное меню FoxPro, где можно вызвать средства отладки и изучить поведение программы или значения ее переменных на этот момент. Здесь следует позаботиться, чтобы возможность прерывания клавишей Escape не была заблокирована в программе командой SET ESCAPE OFF.

Если имеющиеся на терминале окна закрывают фоновый экран, на который произошли нужные выдачи, можно их временно "поднять", нажав клавиши Ctrl-Shift-Alt.

Окна Trace, Debug и Command могут быть перемещены в любое место экрана, и им могут быть приданы любые размеры. Если на экране все-таки не хватает места, можно перейти при отладке в режим плотной выдачи (43-50) строк для адаптеров EGA и VGA с помощью команды SET DISPLAY TO EGA43/MONO43/VGA50.

24.7. Вывод текстовых файлов	252
Глава 25. Средства управления в стиле Windows.....	253
Глава 26. Команда чтения данных Read, многооконный интерфейс.....	265
Глава 27. Команды языка запросов SQL.....	283
Глава 28. Системный интерфейс FoxPro	289
Глава 29. Генераторы приложений	298
29.1. Генератор отчетов	299
29.2. Генератор экранов	312
29.3. Генератор меню.....	321
29.4. Менеджер проектов	328
Глава 30. Средства отладки программ	331
Глава 31. Настройка среды.....	335
Приложение. Системные установки FoxPro	343
Литература	348

Глава 31. НАСТРОЙКА СРЕДЫ

В FoxPro возможны два уровня настройки, которые мы условно назовем внешним и внутренним конфигурированием операционной среды системы.

Внешнее конфигурирование. Внешняя настройка выполняется с помощью специального файла конфигурации системы CONFIG.FP. Это текстовый файл, который программист может заполнить по своему усмотрению. Установки, включаемые в файл CONFIG.FP, начинают действовать сразу после загрузки СУБД в память.

Вот некоторые важнейшие параметры конфигурации:

MVCOUNT — предельно допустимое количество временных переменных. По умолчанию — 256. Диапазон 128...3600 (до 65 000 для расширенной версии FoxPro);

TEDIT — указывает внешний редактор для работы с программными файлами, который будет вызываться вместо собственного редактора по команде MODIFY COMMAND;

WP — указывает внешний редактор для работы с мемо-полями;

F11F12 — указывает, возможно ли в FoxPro обращение к функциональным клавишам F11 и F12. По умолчанию — ON (возможно). Параметр OFF используется в случае, если клавиатура не имеет этих клавиш или чтобы устранить проблему (если она есть) с вводом русской буквы "р".

COMMAND — указывает команду СУБД, которая будет выполнена сразу после ее загрузки. В качестве такой команды может быть, например, задана команда вызова прикладной системы вида

DO <имя командного файла>

В файле CONFIG.FP можно указать и директории/диски для размещения временных файлов системы, программ, оверлейных файлов FoxPro и т.д.

Кроме перечисленных параметров в файл CONFIG.FP могут быть внесены любые команды вида SET, но только без самого слова SET, и между командой и ее параметром ставится знак "=". Поскольку установки SET, включаемые в файл конфигурации, действуют далее всегда (до отмены их в программе или командном окне), целесообразно внести в него лишь самые важные, оставив остальные непосредственно в программах. К таким глобальным установкам можно отнести команды задания цвета (SET COLOR), маршрута поиска файлов (SET PATH), рабочего диска (SET DEFAULT) и, возможно, некоторые другие.

В файле конфигурации могут быть переустановлены любые функциональные клавиши компьютера.

П р и м е р:

```
F11F12=OFF
TEDIT=C:\editor\pe2
DELETED=ON
CONFIRM=OFF
COMMAND=DO kadr
```

Здесь отключено действие клавиш F11 и F12, указан текстовый редактор — PE2.EXE из директории C:\EDITOR. Кроме того, при работе СУБД будут игнорироваться записи, помеченные к удалению (DELETED=ON), и устанавливается необходимость подтверждения завершения редактирования данных в областях GET нажатием клавиши Enter (CONFIRM=OFF). После загрузки СУБД ее работа начинается с выполнения программы KADR, находящейся в текущей директории.

Включение параметра COMMAND может быть удобным в некоторых

случаях как для программиста, так и для пользователя.

Обычно каждая прикладная система хранится на жестком диске в своей отдельной директории, а FoxPro — в своей. Тогда в пользовательской директории можно разместить и свой файл CONFIG.FP, который установит нужную операционную среду (СУБД должна вызываться из данной директории). В этом случае файл CONFIG.FP будет сначала разыскиваться именно в ней и, если его там не оказалось, файл будет загружаться из собственной директории СУБД. Такое размещение обеспечит одновременно с загрузкой FoxPro и автоматический вызов прикладной системы.

При работе с FoxPro в режиме отладки программ удобно сделать некоторые назначения на клавиши. Полезно, например, придать правой (свободной) кнопке мыши функции клавиши Enter. Тогда программист для повторения любой команды (обычно команды DO), введенной им ранее через окно Command, может быстро найти ее с помощью мыши и, не отнимая руки, инициировать, нажав правую кнопку. Кроме того, удобно соединить в одно действие сохранение отлаживаемого PRG-файла (нажатие клавиши Ctrl-W) и его исполнение (ввод команды DO <имя PRG-файла>). Это позволит сэкономить время при отладке, так как указанные действия (редактирование-исполнение, редактирование-исполнение, и т.д.) выполняются программистом практически непрерывно. Ниже приведены команды, закрепляющие за клавишами (правой кнопкой мыши и комбинацией Ctrl плюс клавиша с левой стрелкой) указанные функции:

```
ON KEY LABEL rightmouse KEYBOARD '{enter}'
ON KEY LABEL ctrl+leftarrow KEYBOARD IIF(RIGHT(WONTOP(),3)=;
'PRG','{ctrl+w}{ctrl+f2} DO '+ WONTOP() +' '{enter}','','')
```

В последнем случае сначала анализируется имя текущего окна (WONTOP()). Если оно заканчивается на буквы PRG (это возможно только при загрузке в окно редактора программного файла командой MODIFY COMMAND), буфер клавиатуры заполняется кодом нажатия клавиш Ctrl-W, кодом активации командного окна Ctrl-f2 и командой выполнения (DO) редактируемой программы (WONTOP()), а затем — и кодом клавиши Enter. Таким образом, нажатие клавиш Ctrl-левая_стрелка позволит сразу запомнить программный файл и инициировать его исполнение. Указанные команды нужно поместить в некоторый программный файл, например с именем START.PRG, а его вызов включить в файл CONFIG.FP (строка COMMAND=DO start). Комбинация Ctrl-leftarrow выбрана в виду близости этих двух клавиш на клавиатуре, чтобы нажатие могло быть выполнено только одной правой рукой. Удобно назначить и какую-нибудь одну клавишу, например F12.

Сделанные закрепления будут, естественно, действовать только до отмены, например по команде ON KEY. Если это произошло внутри некоторой прикладной программы, всегда можно выполнить восстановление, введя команду DO start.

Если же вы хотите гарантировать сохранность сделанных назначений, можно включить в системное меню пункт запуска редактируемой программы, как это показано ниже (модуль START.PRG).

```
*-----Программа START.PRG-----
CLEAR
CLEAR MACROS
ON KEY LABEL rightmouse KEYBOARD '{enter}'
* Дополнение системного меню пунктом "Старт-F12"
DEFINE PAD start OF _msysmenu PROMPT "Старт-F12" BEFORE;
  _MSYSTEM SKIP FOR RIGHT(WONTOP(),3)='PRG' KEY f12
ON SELECTION PAD start OF _msysmenu;
  KEYBOARD '{ctrl+w} {ctrl+f2} DO '+ WONTOP() +' '{enter}'
```



```

* Удаление некоторых элементов системного меню:
RELEASE PAD _MSM_DATA OF _msysmenu && PAD-пункта "Database"
RELEASE PAD _MSM_RECROD OF _msysmenu && PAD-пункта "Record"
RELEASE BAR _MST_ABOUT OF _msystem && BAR-пункта "About FoxPro"
* Назначение клавиш вызова для Файлера (F2) и окон отладчика
DEFINE BAR _MST_FILER OF _msystem PROMPT "\<Filer" KEY F2
DEFINE BAR _MWI_DEBUG OF _mwindow PROMPT "\<Debug" KEY Alt-D
DEFINE BAR _MWI_TRACE OF _mwindow PROMPT "\<Trace" KEY Alt-T
* Дополнение WINDOW-меню пунктами для вывода в 43 и 25 строк
DEFINE BAR 1 OF _mwindow PROMPT "43 строки" KEY Alt-4;
SKIP FOR SROWS()>25
DEFINE BAR 2 OF _mwindow PROMPT "25 строк" KEY Alt-2;
SKIP FOR SROWS()=25
ON SELECTION BAR 1 OF _mwindow SET DISPLAY TO EGA43
ON SELECTION BAR 2 OF _mwindow SET DISPLAY TO EGA25
*Дополнение POPUP-меню выбора активной директории
DEFINE PAD razr OF _msysmenu PROMPT 'Разработки'
ON PAD razr OF _msysmenu ACTIVATE POPUP raz
DEFINE POPUP raz
DEFINE BAR 1 OF raz PROMPT '\<FoxPro'
DEFINE BAR 2 OF raz PROMPT 'Текущая директория'
DEFINE BAR 3 OF raz PROMPT 'Система КАДРЫ'
DEFINE BAR 4 OF raz PROMPT 'Система ПЛАН'
ON SELECTION BAR 1 OF raz SET DEFAULT TO (SYS(2004))
ON SELECTION BAR 2 OF raz WAIT WINDOW SYS(2003) NOWAIT
ON SELECTION BAR 3 OF raz SET DEFAULT TO d:\foxpro2\kadr
ON SELECTION BAR 4 OF raz SET DEFAULT TO d:\foxpro2\plan
*-----Конец модуля-----

```

После вызова FoxPro с левой стороны системного меню добавляется пункт "Старт-F12", который будет доступен только при работе с внутренним редактором, куда загружен PRG-файл. При этом клавиша F12 (или любая другая) остается свободной для назначения на нее процедур/команд внутри программ.

Кроме того, может быть полезно переконфигурировать и само системное меню. Целесообразно удалить из него ненужные программисту PAD-пункты Database и Record, а также некоторые пункты POPUP-меню. Например, из POPUP-меню System можно изъять пункты About FoxPro, Puzzle и т.п. Наоборот, имеет смысл сделать более удобным доступ к окнам отладчика (Debug и Trace), закрепив за ними KEY-клавиши Alt-D и Alt-T. В таком случае вызов любого из этих окон может быть осуществлен сразу нажатием указанных клавиш без розыска их через меню. По аналогичным причинам имеет смысл закрепить за Файлером "быструю" клавишу F2.

При отладке программ на экране может стать очень тесно из-за необходимости вывода еще и окон отладчика. Ввиду этого может понадобиться режим плотной выдачи данных в 43 строки. Такую возможность предоставляет дополнительный пункт WINDOW-меню "43 строки". С тем чтобы иметь возможность быстро вернуться к обычному режиму, здесь же реализован и пункт "25 строк".

Поскольку программист обычно одновременно работает над несколькими прикладными системами, имеет смысл включить в меню FoxPro средства быстрого перехода в любую из директорий, где содержатся его разработки. В программе START.PRG эта возможность реализована с помощью POPUP-меню RAZR ("Разработки"). Здесь пункт FoxPro позволяет вернуться в "родительскую" директорию FoxPro (SET DEFAULT TO (SYS(2004))), пункт "Текущая директория" — выяснить имя текущей директории (SYS(2003)), пункты "Система КАДРЫ" и "Система ПЛАН" — перейти в соответствующие директории (d:\foxpro2\kadr, d:\foxpro2\plan) диска.

Сформированное таким образом системное меню изображено на рис.31.1. Курсор находится в пункте "Разработки".

При определении операционной среды не следует запрашивать число переменных "про запас". Это отнимает ее у прикладной системы, снижая скорость обработки данных. Если возможно, лучше даже уменьшить

запрашиваемые ресурсы относительно принятых по умолчанию.

На работу FoxPro влияют и параметры FILES и BUFFERS в файле конфигурации операционной системы CONFIG.SYS. Параметр FILES должен указывать по крайней мере на 10 файлов больше, чем может быть одновременно открыто в прикладной системе. Это необходимо для работы самой СУБД. Рекомендуемое минимальное число 40. Параметр BUFFERS влияет на скорость обмена данными с диском. Рекомендуемое число 20...40.

Старт-F12 System File Edit Program Window Разработки

FoxPro
Текущая директория
Система КАДРЫ
Система ПЛАН

Рис.31.1

Внутреннее конфигурирование. Внутреннее конфигурирование системы осуществляется с помощью так называемого ресурсного файла, являющегося стандартным файлом базы данных, — FOXUSER.DBF, а также его вспомогательного файла мемо-полей — FOXUSER.FPT. Ресурсный файл содержит информацию о настройке внутреннего редактора, положении и виде BROWSE-окон, цветовых наборах и т.д. Ресурсный файл обновляется автоматически при работе СУБД как в командном окне, так и в программном режиме. Обращение к ресурсному файлу может быть отменено командой

■ SET RESOURCE OFF

(по умолчанию ON). Возможно также создание ресурсного файла с собственным именем с помощью команды

■ SET RESOURCE TO <имя ресурсного файла>

Если вы хотите просмотреть и отредактировать содержимое ресурсного файла, следует сначала отключить его от СУБД, открыть и вызвать, например, в BROWSE-окно, т.е. набрать следующую последовательность команд:

```
.SET RESOURCE OFF
.USE FOXUSER
.BROWSE
```

а по завершении работы с ним выполнить обратные действия:

```
.USE
.SET RESOURCE ON
```

Этот файл имеет следующие поля:

TYPE	— тип информации, содержащейся в записи;
ID	— вид информации;
NAME	— имя объекта, настройки которого запоминаются;
READONLY	— признак возможности изменять эти данные;
CKVAL	— служебная информация;
DATA	— содержательные данные о настройке (мемо-поле);
UPDATED	— дата обновления записи.

Поле ID указывает на объекты, конфигурация которых сохраняется. Важнейшими для нас являются COLORSET, WINDBROW, WINDMODIFY (цветовые установки, конфигурация BROWSE-окна и окна редактора).

Например, возможны такие записи (поля TYPE, ID, NAME):

```
PREF2.0 WINDBROW KADR
```

— установки окна BROWSE для базы KADR.DBF,

```
PREF2.0 WINDMODIFY KADR.PRG
```

— установки внутреннего редактора FoxPro, вызываемого по команде MODIFY COMMAND/FILE для файла KADR.PRG.

Хотя все данные в ресурсном файле доступны для редактирования, изменять разрешается только логическое поле READONLY, которое имеет значение .F.. Его можно изменить на .T., если настройка данного параметра завершена. Далее она изменена быть не может, даже если вы фактически изменили вид объекта.

Другое действие, которое разрешено над ресурсным файлом, — это пометка к удалению ненужных записей и их упаковка.

Концепция ресурсного файла ориентирована как на пользователя, работающего через командное окно, так может быть полезна и программисту.

Ресурсному файлу можно передать описание BROWSE-окна. В этом случае следует выполнить программу, где в команде BROWSE (кроме содержательных параметров) используется опция LAST или PREFERENCE (см. команду BROWSE):

```
BROWSE <параметры> PREFERENCE <имя "установки">
```

После этого описание окна запоминается в ресурсном файле, и команда BROWSE может теперь использоваться без каких-либо параметров — только со ссылкой на <имя "установки">:

```
BROWSE PREFERENCE <имя "установки">
```

которая будет автоматически извлекаться из ресурсного файла.

Кроме того, в ресурсном файле можно сохранять и оттуда извлекать цветовые наборы (команда SET COLOR SET TO), а также конфигурацию окна редактора (пункты системного меню EDIT+PREFERENCE).

Если необходимости в ресурсном файле нет, с целью экономии времени лучше отключить его командой SET RESOURCE OFF.

Адаптирование FoxPro для русскоязычного пользователя. Если вы работаете с оригинальным пакетом FoxPro, то сразу обратите внимание на то, что СУБД не позволяет осуществлять ввод русской буквы "Н" и, возможно, буквы "р".

"Восстановление" буквы "р" может быть выполнено очень легко. Для этого следует просто загрузить FoxPro с ключом "-k", т. е.

```
FOXPRO -k
```

либо ввести в файл CONFIG.FP следующую строку:

```
F11F12=OFF
```

При этом, правда, системе перестанут быть доступны функциональные клавиши F11 и F12. Но потеря невелика.

С буквой Н дело обстоит сложнее. Для решения проблемы придется вовсе отказаться от русской Н и заменить ее на латинскую. Эта замена (в драйвере кириллицы для клавиатуры) может быть легко произведена каждым пользователем с помощью любого редактора. После этого при нажатии на клавишу русской Н будет генерироваться латинская Н. Однако такая замена приводит к тому, что индексирование по символьным полям даст искаженную последовательность следования букв. Вначале будут идти слова, начинающиеся на Н (она имеет самый меньший код), а только потом — на А, Б, В и т.д. Если это

имеет значение, индексирование должно выполняться с учетом указанного фактора.

Стандартное решение вопроса может быть осуществлено с помощью функции SYS(15,<вырС1>,<вырС2>), которая заменяет символы из <вырС2> на символы из <вырС1>. При этом скорость индексирования снижается приблизительно на треть.

В комплекте поставки FoxPro имеется специальная перекодировочная таблица, хранящаяся в переменной EUROPEAN, содержащейся в файле EUROPEAN.MEM. Чтобы ею воспользоваться, необходимо изменить ее содержимое. Лучше изменить и имя переменной на какое-то другое, например RUS, и хранить ее в файле RUS.MEM. Для этого следует загрузить файл EUROPEAN.MEM, занести переменную EUROPEAN в переменную RUS и сохранить ее в файле RUS.MEM, как показано ниже.

```
.RESTORE FROM european  
.rus=european  
.SAVE ALL LIKE rus TO rus
```

Затем нужно загрузить файл RUS.MEM в какой-нибудь редактор (но не в редактор FoxPro). Проще всего это сделать прямо в файловом мониторе Norton Commander, нажав клавишу F4 для файла RUS.MEM. Мы увидим на экране редактора файл RUS.MEM, содержащий переменную RUS, пока совпадающую с переменной EUROPEAN. Эта переменная включает практически все символы ASCII-таблицы. Первую половину таблицы (вплоть до символа с кодом 127) оставляем без изменения, за исключением латинской буквы H, которую заменим на русскую букву Н. Если вы уже изменили драйвер клавиатуры, то русскую Н следует ввести ее кодом, т.е. набрать Alt-141. Во вторую половину, предназначенную для символов национальных алфавитов и символов псевдографики, последовательно вносим буквы "АБВГДЕЖЗИЙКЛМ" и латинскую букву H. На этом ввод нужно прекратить и остальные символы справа удалить.

В процессе индексирования, встречая в индексируемом поле базы данных перечисленные буквы из второй половины таблицы, FoxPro в индексном файле замещает их кодами из первой половины, в частности, латинская H будет заменена на русскую Н.

Для индексирования базы данных с использованием функции SYS(15) и работы с ней в дальнейшем необходимо постоянно иметь в памяти переменную RUS, которая предварительно загружается командой RESTORE. В следующем примере проиндексируем базу данных KADR.DBF по полю FAM с созданием файла KADRFAM1.IDX:

```
.RESTORE FROM rus  
.INDEX ON SYS(15,rus,fam) TO kadrfam1  
.LIST fam
```

Тогда, если в поле FAM имеются фамилии, начинающиеся на букву H, они будут стоять в индексе после буквы М.

Однако ускоренный поиск фамилий, начинающихся на H, по команде SEEK теперь будет невозможен, поскольку с клавиатуры мы можем ввести только латинскую букву H, а в индексе стоит русская. Отсюда следует, что и при задании ключа поиска нужно использовать функцию SYS(15):

```
SEEK SYS(15,rus,<ключевое выражение>)
```

Например:

```
SEEK SYS(15,rus,'НИКОЛАЕВ')
```

Вследствие того что для русскоязычного пользователя рассмотренная замена

букв изменила положение только буквы Н, можно предложить и другое решение — с применением функций CHRTRAN() или STRTRAN() (см. гл.16 "Функции СУБД"), которыми заменим латинскую Н на русскую Н при индексировании и поиске:

```
INDEX ON CHRTRAN(<индексируемое поле>,'Н',CHR(141)) TO <индекс>  
SEEK CHRTRAN(<ключевое выражение>,'Н',CHR(141))
```

Здесь Н — латинская буква; CHR(141) — русская буква Н (ее код 141).

Хотя такое решение проще и кажется предпочтительнее из-за того, что не требуется затрат памяти для хранения переменной RUS, анализ временных характеристик показывает, что оно не самое лучшее. По сравнению с использованием функции SYS(15) индексирование с применением функции CHRTRAN() выполняется на 10...15%, а с применением STRTRAN() — до (в худшем случае) 3—5 раз медленнее.

В заключение отметим, что есть отечественные русифицированные версии пакета FoxPro, которые позволяют избежать указанных проблем, а также реализуют большинство из строковых функций конвертирования букв (заглавных в строчные и наоборот).

Исполнение готовых программ. Готовые программы в форме FXP- и APP-файлов могут выполняются самой системой FoxPro и вместе с ней устанавливаться у заказчика. Однако фирма-разработчик возражает против такой несанкционированной передачи пакета FoxPro и для распространения готовых программ предлагает приобретать специальный дистрибьюторский пакет (Disrtibutin Kit), который содержит файлы только для исполнения готовых программ (без возможности разработки), а также компилятор для генерации исполняемых EXE-файлов.

С помощью этого пакета можно распространять готовые прикладные системы в одном из следующих трех видов.

1. FXP- или APP-файлы вместе со средствами их исполнения.
2. Компактные EXE-файлы со средствами их исполнения.
3. Независимые исполнимые EXE-файлы. Такие файлы являются самодостаточным представлением программ и для своего исполнения не нуждаются в каких-либо интерпретаторах. Размер независимого EXE-файла может быть весьма значительным.

APP- и EXE-файлы обычно создаются с помощью Менеджера проектов (см. разд.29.4).

Сейчас рассмотрим только некоторые вопросы исполнения программ.

Готовые программы в форме FXP- и APP-файлов выполняются в присутствии файлов библиотеки поддержки:

```
FOX.R.EXE, FOXPRO.ESL, FOXPRO.ESO,
```

а программы в форме компактных EXE-файлов — при наличии файлов

```
FOXPRO.ESL и FOXPRO.ESO.
```

Для загрузки готовой FXP/APP-программы следует воспользоваться динамическим загрузчиком (файл FOXR.EXE) в форме команды на уровне DOS. Здесь же, если нужно, можно указать передаваемые ключи (-C, -E, -K, -T), поместив их непосредственно перед именем программного файла:

```
FOXR [<ключи>] <имя программного файла>
```

Например, команда

```
FOXR -T kadr
```


загружает программу KADR.FXP/APP и при этом ключом -T подавляется вывод заглавного кадра FoxPro (демонстрация торговой марки). Чтобы упростить загрузку, такую строку удобно поместить в некоторый командный BAT-файл.

Вместе с тем указание ключей и имени программы возможно и другим способом. Ключи для FXP/APP- и компактных EXE-файлов могут быть заданы в переменной среды DOS с именем FOXPROSWX, которая создается с помощью команды DOS SET и помещается в файл AUTOEXEC.BAT. Например,

```
FOXPROSWX=-T
```

Имя программы может быть помещено в файл конфигурации CONFIG.FP в виде строки

```
COMMAND = DO <имя программы>
```

Компактные и автономные EXE-файлы загружаются просто их вызовом в командной строке DOS. Необходимые ключи передаются в программу в форме автономного EXE-файла путем их перечисления после имени файла.

Готовый программный продукт в любой из указанных форм поддерживает все команды и функции FoxPro (в том числе большую часть системного меню), за исключением, естественно, тех, которые никогда не включаются в программу, например команд генератора приложений, отладки, компиляции и т.п.

Приложение. СИСТЕМНЫЕ УСТАНОВКИ FoxPro

Таблица 1. Параметры команд SET (начало).

SET-команда	Значение параметра	По умолчанию
ALTERNATE	<имя файла>	
ALTERNATE	OFF/ON	OFF
ANSI	OFF/ON	OFF
AUTOSAVE	OFF/ON	OFF
BELL	OFF/ON	ON
BELL	19-10000 Г. и 2-19 с. (частота и продолжит.)	512, 2
BLINK	ON/OFF	ON
BLOCKSIZE	<вырN>	64
BORDER	<атрибут>	SINGLE
BRSTATUS	OFF/ON	OFF
CARRY	OFF/ON	OFF
CENTURY	OFF/ON	OFF
CLEAR	ON/OFF	ON
CLOCK	OFF/ON	OFF
CLOCK	<координаты>	0; 69
COLOR	<цветовые атрибуты>	
COLOR OF BOX	<цветовые атрибуты>	
COLOR OF FIELDS	<цветовые атрибуты>	
COLOR OF HIGHLIGHT	<цветовые атрибуты>	
COLOR OF INFORMATION	<цветовые атрибуты>	
COLOR OF NORMAL	<цветовые атрибуты>	
COLOR OF MESSAGES	<цветовые атрибуты>	
COLOR OF TITLES	<цветовые атрибуты>	
COLOR OF SCHEME	<список цветовых пар>	Текущие установки
COLOR SET	<имя цветового набора>	
COMPATIBLE	OFF/ON (FOXPLUS/DB4)	OFF (FOXPLUS)
CONFIRM	OFF/ON	OFF
CONSOLE	ON/OFF	ON
CURRENCY	<денежный символ>	"\$"
CURRENCY	<позиция>	LEFT
CURSOR	ON/OFF	ON
DATE	<формат даты>	AMERICAN
DEBUG	ON/OFF	ON
DECIMALS	<от 0 до 18>	2
DEFAULT	<диск/каталог>	
DELETED	OFF/ON	OFF
DELIMITERS	OFF/ON	OFF
DELIMITERS	<вырC>/DEFAULT	"."
DEVELOPMENT	ON/OFF	ON
DEVICE	SCREEN/PRINT /FILE<файл>	SCREEN
DISPLAY	<тип дисплея>	Установленный
ECHO	OFF/ON	OFF
ESCAPE	OFF/ON	ON
EXACT	ON/OFF	ON
EXCLUSIVE	ON/OFF	ON
FULLPATH	ON/OFF	ON
F<число>	<строка>	
HEADING	ON/OFF	ON
HELP	ON/OFF	ON
HELP	<имя файла>	FOXHELP
HOURS	12/24	12
INTENSITY	ON/OFF	ON
LOGERROR	ON/OFF	ON
MACKEY	<клавиша>	F10
MARGIN	<от 0 до 254>	0
MARK	<символ>	" / "
MEMOWIDTH	<от 8 до 32000>	50
MOUSE	<от 1 до 10>	5
MOUSE	OFF/ON	ON
NEAR	OFF/ON	OFF
NOTIFY	OFF/ON	ON
ODOMETER	<от 1 до 32767>	100

Таблица 1. Параметры команд SET (окончание).

SET-команда	Значение параметра	По умолчанию
OPTIMIZE	OFF/ON	ON
PATH	<маршрут>	" "
POINT	<символ>	" "
PRINT	ON/OFF	ON
RESOURCE	ON/OFF	ON
RESOURCE	<имя файла>	FOXUSER
SAFETY	ON/OFF	ON
SCOREBOARD	OFF/ON	OFF
SEPARATOR	<символ>	" "
SPACE	ON/OFF	ON
STATUS	OFF/ON	OFF
STEP	OFF/ON	OFF
STICKY	ON/OFF	ON
SYSMENU	ON/OFF/AUTOMATIC	ON
TALK	ON/OFF	ON
TABS	<строка>	нулевая строка
TEXTMERGE	OFF/ON	OFF
TRBETWEEN	ON/OFF	ON
TYPEAHEAD	<от 0 до 32000>	20
UDFPARMS	VALUE/REFERENCE	VALUE
UNIQUE	OFF/ON	OFF

Таблица 2. Специальные установки файла CONFIG.FP

Установки	Значение параметра	Принято по умолчанию
COMMAND	<команда>	OFF
DOSMEM	ON/OFF <вырN>	OFF
EDITWORK	<каталог>	стартовая директория
EMS	ON/OFF <вырN>	ON
EMS64	ON/OFF	ON
F11F12	ON/OFF	ON
_GENGRAPH	<имя программы>	GENGRAPH.PRG
_GENMENU	<имя программы>	GENMENU.PRG
_GENPD	<имя программы>	GENPD.APP
_GENSCRN	<имя программы>	GENSCRN.PRG
_GENXTAB	<имя программы>	GENXTAB.PRG
INDEX	<расширение>	IDX
LABEL	<расширение>	LBX
MVCOUNT	<от 128 до 3600> <от 128 до 65000> для FoxPro (X)	256
OUTSHOW	ON/OFF	ON
OVERLAY	<каталог>[OVERWRITE]	каталог FoxPro
PROGWORK	<каталог>	стартовая директория
REPORT	<расширение>	FRX
RESOURCE	<имя маршрута>	FOXUSER или стартовая директория
SORTWORK	<каталог>	стартовая директория
TEDIT	[/<вырN><редактор>	
TIME	<от 1 до 1000000>	6000
TMPFILES	<дисковод:>	стартовая директория

Таблица 3. Начальные установки функциональных клавиш

Клавиша	Значение
F1	HELP; - вызов окна оперативной помощи Help
F2	SET; - вызов окна View для открытия баз, установления связей, задания SET-установок
F3	LIST; - вывод записей активной базы данных
F4	DIR; - вывод имен файлов из текущей директории
F5	DISPLAY STRUCTURE; - вывод структуры активной базы
F6	DISPLAY STATUS; - вывод информации состоянии СУБД
F7	DISPLAY MEMORY; - вывод всех временных переменных
F8	DISPLAY; - вывод записей активной базы с паузами
F9	APPEND; - вызов команды дополнения базы данных
F10	Активация/деактивация главного меню системы.
F11-F12	Свободные клавиши

Точка с запятой в конце каждой команды генерирует символ возврата каретки Enter, что обеспечивает немедленное выполнение команды. Без точки с запятой символьные строки будут только выводиться. За F-клавишами с помощью команды SET FUNCTION можно закрепить и другие символьные строки длиной до 254 символов.

Таблица 4. Закрепления цветовых схем

NN схем	Название	Объекты FoxPro
1	User Winds	Окна, определенные пользователем
2	User Menus	Меню, определенные пользователем
3	Menu Bar	Горизонтальная строка системного меню
4	Menu Pops	Системные POPUP-меню
5	Dialogs	Системные диалоги
6	Dialog Pops	POPUP-меню в системных диалогах
7	Alerts	Системные предупреждающие сообщения
8	Windows	Системные окна
9	Window Pops	POPUP-меню в системных окнах
10	Browse	Окна Browse
11	Report	Окно генератора отчетов
12	Alert Pops	Области редактирования и POPUP в диалогах
13-16		Резерв для последующего использования
17-24		Схемы пользователей

Таблица 5. Цветовые пары системных цветовых схем

Схе- мы	Цветовые пары									
	1	2	3	4	5	6	7	8	9	10
1	W+/B	W+/BG	GR+/B	GR+/B	R+/B	W+/GR	GR+/RB	N+/N	GR+/B	R+/B
2, 4	BG/W	N/W	N/W	B/W	W/N	N/BG	W+/W	N+/N	B/W	W/N
3	BG/W	N/W	BG/N	BG/N	BG/N	N/BG	W+/W	N+/N	BG/N	BG/N
5	W+/RB	W+/BG	W+/RB	W+/RB	W/RB	W+/B	GR+/RB	N+/N	W+/RB	W/RB
6	W/BG	W+/BG	W+/RB	W+/RB	W/RB	W+/B	BG+/BG	N+/N	W+/RB	W/RB
7, 12	GR+/R	W+/W	GR+/R	W+/R	W/R	W+/N	GR+/R	N+/N	W+/R	W/R
8	W+/BG	W+/W	GR+/W	GR+/W	N+/W	W+/GR	BG+/BG	N+/N	B/BG	W/BG
9	W/BG	W+/BG	B/BG	GR+/W	N+/W	W+/GR	W+/B	N+/N	GR+/W	N+/W
10	W+/BG	GR+/B	GR+/W	GR+/W	N+/W	GR+/GR	W+/B	N+/N	GR+/W	N+/W
11	W+/BG	W+/W	GR+/W	GR+/W	N+/W	W+/GR	W/B	N+/N	W+/B	W/BG

Остальные цветовые схемы (13-24) совпадают со схемой 1.

Важными для программиста являются схемы:

1 — Окна пользователя (User Windows)

2 — Меню пользователя (User Menus)

4 — Вертикальное меню (Menu Pops)

10 — Browse-окно.

Цветовые схемы 2 и 4 (их цветовые пары совпадают) применяются для раскраски всех видов пользовательских меню.

Таблица 6. Окраска элементов окон, меню и BROWSE-окон

1. Окна пользователя	2, 4. Меню	10. BROWSE-окно
1 поле SAY	Недоступные опции	Другие записи
2 поле GET	Доступные опции	Текущее поле
3 Рамка	Рамка	Рамка
4 Активный заголовок	Заголовок меню	Активный заголовок
5 Неактивный заголовок и сообщение	Сообщение	Неактивный заголовок
6 Выбранная опция	Выбранная опция	Выбранный текст
7 Часы, горячие клавиши	Горячие клавиши	Текущая запись
8	Тень	.
9	Доступный ключ управления	.
10	Недоступный ключ управления	.

Пометка записи, предназначенной для удаления в BROWSE/CHANGE-окне, делается цветом фона из пары 7, а именно /B.

Имена элементов системного меню _MSYSMENU

Ниже указаны приглашения и имена всех PAD-пунктов системного меню FoxPro, а под ними приглашения и имена всех строк соответствующих POPUP-меню. Кроме того — пользователю программисту доступны имена некоторых важнейших POPUP-меню более низких уровней. Они перечислены в конце.

System _MSYSTEM

About FoxPro	_MST_ABOUT
Help...	_MST_HELP
Macros	_MST_MACRO
	_MST_SP100
Filer	_MST_FILER
Calculator	_MST_CALCUL
Calendar/Diary	_MST_DIARY
Special Character	_MST_SPECI
ASCII Chart	_MST_ASCII
Capture	_MST_CAPTR
Puzzle	_MST_PUZZL

File _MFILE

New...	_MFI_NEW
Open...	_MFI_OPEN
Close	_MFI_CLOSE
Close All	_MFI_CLALL
	_MFI_SP100
Save	_MFI_SAVE
Save as...	_MFI_SAVAS
Revert	_MFI_REVRT
	_MFI_SP200
Printer Setup	_MFI_SETUP
Print...	_MFI_PRINT
	_MFI_SP300
Quit	_MFI_QUIT

Edit _MEDIT

Undo	_MED_UNDO
Redo	_MED_REDO
	_MED_SP100
Cut	_MED_CUT
Copy	_MED_COPY
Paste	_MED_PASTE
Clear	_MED_CLEAR
	_MED_SP200
Select All	_MED_SLCTA
	_MED_SP300
Goto Line...	_MED_GOTO
Find...	_MED_FIND
Find Again	_MED_FINDA
Replace and	_MED_REPL
Replace All	_MED_REPLA
	_MED_SP400
Preferences	_MED_PREF

Database _MDATA

Setup...	_MDA_SETUP
Browse	_MDA_BROW
	_MDA_SP100
Append From	_MDA_APPND
Copy To...	_MDA_COPY
Sort...	_MDA_SORT
Total...	_MDA_TOTAL
	_MDA_SP200
Average...	_MDA_AVG
Count...	_MDA_COUNT
Sum	_MDA_SUM
Calculate...	_MDA_CALC
Report...	_MDA_REPR
Label...	_MDA_LABEL
	_MDA_SP300
Pack	_MDA_PACK
Reindex	_MDA_RINDEX

Record _MRECORD

Append	_MRC_APPND
Change	_MRC_CHNGE
	_MRC_SP100
Goto...	_MRC_GOTO
Locate	_MRC_LOCAT
Continue	_MRC_CONT
Seek	_MRC_CONT
	_MRC_SP200
Replace	_MRC_REPL
Delete...	_MRC_DELET
Recall...	_MRC_RECAL

Program _MPROG

Do...	_MPG_DO
	_MPR_SP100
Cancel	_MPR_CANSL
Resume	_MPR_RESUM
	_MPR_SP200
Compile...	_MPG_COMPL
Generate...	_MPG_GENER
FoxDoc	_MPR_DOCUM
FoxGraph...	_MPR_GRAPH

Window	_MWINDOW
Hide	MWI_HIDE
Hide All	MWI_HIDEA
Show All	MWI_SHOWA
Clear	MWI_CLEAR
	MWI_SP100
Move	MWI_MOVE
Size	MWI_SIZE
Zoom	MWI_ZOOM
Zoom	MWI_MIN
Cycle	MWI_ROTAT
Color...	MWI_COLOR
	MWI_SP200
Command	MWI_CMD
Debug	MWI_DEBUG
Trace	MWI_TRACE
View	MWI_VIEW

Имена вспомогательных POPUP-меню

MREPORT	- меню генератора отчетов
MLABEL	- меню генератора этикеток
MBROWSE	- меню BROWSE-окна
MMACRO	- меню создания макросов
MDIARY	- меню доступа к дневнику
MFILER	- меню файлового процессора
MSCREEN	- меню генератора экранов
MMBLDR	- меню генератора меню
MPROJ	- меню генератора проектов
MRQBE	- меню генератора запросов

Список литературы

1. Берещанский Д.Г. Практическое программирование на dBASE. — М.: Финансы и статистика, 1989. — 192 с.
2. Гринберг Ф., Гринберг Р. Самоучитель программирования на входном языке СУБД dBASEIII. — М.: Мир, 1989, — 453 с.
3. Дейт К. Руководство по реляционной СУБД DB2. — М.: Финансы и статистика, 1988. — 320 с.
4. Каррабис Д.Д. Программирование в dBASEIII Plus. — М.: Финансы и статистика, 1991. — 249 с.
5. Крамм Р. Системы управления базами данных dBASEII и dBASEIII для персональных компьютеров. — М.: Финансы и статистика, 1988. — 283 с.
6. Программное обеспечение персональных ЭВМ. Справочное пособие/ Под ред. А.А. Стогния — Киев: Наукова думка, 1989. — 361 с.
7. Романов Б.А., Кушниренко А.С. dBASEIV. Назначение, функции, применение. — М.: Радио и связь, 1991. — 384 с.
8. FoxPro-2.0. Interface Guide. — MicroSoft Corp., 1992.
9. FoxPro-2.0. Developer's Guide. — MicroSoft Corp., 1992.
10. FoxPro-2.0. Commands & Functions. — MicroSoft Corp., 1992.
11. Miriam Liskin. Programming FoxPro-2.0. — Ziff-Davis Press, 1992. — 1116 p.
12. H.Ahlo a.o. FoxPró 2: A Developer's Guide. — M&T Books, 1992.
13. Michael Antonovich. FoxPro 2. Programming Guide. — Microtrend Books, 1992.
14. James E. Powell. FoxPro 2. Developer's Library. — Microtrend Books, 1992.
15. Bill Chambers. FoxPro 2. Business & MIS Application. — Microtrend Books, 1992.

Оглавление

Введение.....	3
Глава 1. Технические характеристики и особенности СУБД	7
Глава 2. Обозначения и структура команд СУБД.....	14
Глава 3. Создание файла базы данных.....	18
3.1. Создание структуры файла.....	18
3.2. Заполнение базы данных.....	20
Глава 4. Окно редактирования.....	23
4.1. BROWSE-окно.....	25
4.2. CHANGE/EDIT-окно	32
Глава 5. Перемещения в базе данных	34
Глава 6. Просмотр данных	35
Глава 7. Удаление данных.....	36
Глава 8. Изменение данных	37
Глава 9. Локализация и поиск данных в базе	39
9.1. Фильтрация данных.....	39
9.2. Последовательный поиск	39
9.3. Индексирование баз данных.....	40
9.4. Технология Rushmore.....	52
Глава 10. Работа с несколькими базами	54
Глава 11. Создание командных файлов	58
Глава 12. Команды ввода-вывода.....	62
Глава 13. Работа с переменными	76
Глава 14. Команды управления.....	78
Глава 15. Организация циклов.....	79
Глава 16. Функции СУБД.....	82
Глава 17. Массивы переменных	105
Глава 18. Организация меню в прикладных системах.....	110
18.1. Световое меню	110
18.2. Клавишное меню.....	135
18.3. Создание меню с помощью функций	138
18.4. Использование функциональных клавиш	139
18.5. Дополнительные вопросы.....	140
Глава 19. Изобразительные средства СУБД	150
Глава 20. Работа с окнами	158
Глава 21. Модульность программ.....	166
Глава 22. Примеры программ в среде СУБД FoxPro	172
22.1. Поддержка базы данных	172
22.2. Древовидная организация данных.....	196
22.3. Работа со словарями	213
22.4. Пример системы обработки данных	219
Глава 23. Надежность систем обработки данных	228
Глава 24. Манипулирование файлами	237
24.1. Копирование файлов.....	237
24.2. Работа со структурами БД.....	239
24.3. Взаимодействие файлов	244
24.4. Сортировка данных	246
24.5. Математическая обработка БД.....	247
24.6. Работа с мемо-полями.....	250

24.7. Вывод текстовых файлов	252
Глава 25. Средства управления в стиле Windows.....	253
Глава 26. Команда чтения данных Read, многооконный интерфейс.....	265
Глава 27. Команды языка запросов SQL.....	283
Глава 28. Системный интерфейс FoxPro	289
Глава 29. Генераторы приложений	298
29.1. Генератор отчетов	299
29.2. Генератор экранов.....	312
29.3. Генератор меню.....	321
29.4. Менеджер проектов.....	328
Глава 30. Средства отладки программ	331
Глава 31. Настройка среды.....	335
Приложение. Системные установки FoxPro	343
Литература	348

214333

Производственное издание

ПОПОВ Александр Алексеевич

Программирование в среде СУБД FoxPro 2.0

Заведующий редакцией Ю. Г. Ивашов

Редакторы С. Н. Удалова, В. И. Ченцова

Обложка художника Н. И. Казакова

Художественный редактор В. И. Мусиенко

Технический редактор Т. Г. Тертышная

ЛР № 010164 от 04.01.92

Подписано в печать с оригинал-макета 27.07.93 Формат 70×100/16 Бумага тип. № 2 Гарнитура Таймс Печать офсетная

Усл.печ.л. 28,60 Усл.кр.-отт. 28,93

Уч.-изд.л. 28,0 Тираж 30 000 экз. Изд. № 23792 Зак. № 279

С-082

Издательство "Радио и связь". 101000 Москва, Почтамт, а/я 693

Московская типография № 4 Министерства печати и информации РФ. 129041 Москва, Б.Переяславская, 46

FLEXIS II

ГИПЕР
ТЕКСТОВАЯ
СУБД
VER. 2.20

Система FLEXIS II - удобный инструмент для создания компьютерных энциклопедий, справочных баз данных, электронных журналов, архивов, каталогов и презентаций.

Информация в базе данных хранится в виде гипертекста. Гипертекст можно определить как набор информационных объектов, связанных между собой в единую систему. Каждый объект представляется на экране, пользователь может по контексту выбрать одну из связей и перейти к другой информационной вершине. Таким образом можно построить базу данных, отражающую сложные взаимодействия между объектами реального мира, не используя сложных математических моделей.

СУБД FLEXIS II позволяет вносить в базу данных не только текстовую и формализованную табличную информацию, но и объединять в единой системе графические, звуковые, бинарные и специальные пользовательские типы данных, а также подключать в виде объектов базы данных внешние программы. FLEXIS II поддерживает прямую и обратную навигацию по гипертекстовой БД, позволяет импортировать и экспортировать данные.

Уникальная простота работы с СУБД FLEXIS II позволяет работать с системой непосредственно информационным работникам и экспертам, не владеющим навыками программирования. Минимум компьютерных знаний - основы программы Norton Commander и любого текстового редактора. И это все!

Программа предназначена для IBM-совместимых компьютеров - от XT до PS/2 и AT486. Общее количество записей в БД - до 80 тысяч, объемы БД - десятки мегабайт.



109044, Москва, ул. Крутицкая, 9.
Тел. (095) 276-31-97, 276-40-57.

А. А. ПОПОВ